



# Intel® Integrated Performance Primitives for Intel® Architecture

---

*Quick Reference*

## **Part 2: Image and Video Processing**

Document Number: 254145-006US

World Wide Web: <http://developer.intel.com>

<b>Version</b>	<b>Version Information</b>	<b>Date</b>
-001	Original issue. Documents API included into Intel IPP release 4.0 gold. Links to Intel IPP Reference Manual (document number A70805-4002).	10/2003
-002	Documents API included into Intel IPP release 4.1 beta. Links to Intel IPP Reference Manual (document number A70805-013).	05/2004
-003	Documents API included into Intel IPP release 4.1. Links to Intel IPP Reference Manual (document number A70805-014).	07/2004
-004	Documents API included into Intel IPP release 5.0 beta. Links to Intel IPP Reference Manual (document number A70805-015)	04/2005
-005	Documents API included into Intel IPP release 5.0. Links to Intel IPP Reference Manual (document number A70805-016)	08/2005
-006	Documents API included into Intel IPP release 5.1. Links to Intel IPP Reference Manual (document number A70805-017)	03/2006

The information in this manual is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Intel, the Intel logo, Intel SpeedStep, Intel NetBurst, Intel NetStructure, MMX, Intel386, Intel486, Celeron, Intel Centrino, Intel Xeon, Intel XScale, Itanium, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other names and brands may be claimed as the property of others..

Copyright © 2005-2006 Intel Corporation.

# Overview

---

This Quick Reference provides a full list of prototypes for all functions included into the image and video processing domain of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® architecture.

The Intel IPP software is a new generation of the Intel® Performance Libraries, comprising a broad range of functions for basic software functionality and including, among many others, the image and video processing functions domain.

Intel IPP is a cross architecture software library optimized for the latest generations of Intel microprocessors, including Intel® Pentium® 4 processor, Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3), Intel® Xeon™ processor, Intel® Xeon™ processor with Intel® Extended Memory 64 Technology (Intel® EM64T), Intel® Itanium® 2 processor, and Intel® PCA application processors.

The image and video processing subset of Intel IPP includes the following functional domains:

- [Support Functions](#)
- [Image Data Exchange and Initialization Functions](#)
- [Image Arithmetic and Logical Operations](#)
- [Image Color Conversion](#)
- [Threshold and Compare Operations](#)
- [Morphological Operations](#)
- [Filtering Functions](#)
- [Image Linear Transforms](#)
- [Image Statistics Functions](#)
- [Image Geometric Transforms](#)

- [Wavelet Transforms](#)
- [Computer Vision](#)
- [Image Compression Functions](#)
- [Video Coding](#)

This Quick Reference contains an alphabetic list of all image and video processing function names with hyperlinks to the full set of function prototypes available in the library and a short description of function operation. The detailed information about all functions, including introductory material, general argument description, function behavior explanation, return values, examples and more, is given in the Intel® Integrated Performance Primitives Reference Manual, volume 2 (document number A70805-017).

Each section and function name included into the reference section of this Quick Reference has a hyperlink to the respective section of the above Reference Manual.

For these links to work properly, please make sure that:

1. You have the correct version of the Intel IPP Reference manual available with the Intel IPP distribution (file name ippiman.pdf), and
2. Files for the Quick Reference and for the Reference Manual reside in the same directory.

If you have jumped to the manual and wish to return to the Quick Reference, use the Go to the Previous View button in the Adobe Acrobat\* tool. If you plan to jump between the Quick Reference and the Manual frequently, it is preferable to open the Reference Manual before clicking hyperlinks in the Quick Reference. In this case you will see both the documents in the Window menu of your viewer.

For more information about the Intel IPP library, please refer to the Web site at [developer.intel.com/software/products/ipp/](http://developer.intel.com/software/products/ipp/).

## Alphabetic List of Routines

### A

Abs  
AbsDiff  
AbsDiffC  
Add  
Add128\_JPEG  
Add8x8  
Add8x8HP  
AddBackPredPB\_H263  
AddC  
AddC8x8  
AddProduct  
AddRandGauss\_Direct  
AddRandUniform\_Direct  
AddRotateShift  
AddSquare  
AddWeighted  
AlphaComp  
AlphaCompC  
AlphaPremul  
AlphaPremulC  
And  
AndC  
ApplyHaarClassifier  
ApplyMixedHaarClassifier  
Average8x8,  
Average16x16

### B

BGR565ToYCbCr\_JPEG,  
BGR555ToYCbCr\_JPEG  
BGR565ToYCbCr411,  
BGR555ToYCbCr411  
BGR565ToYCbCr411LS\_MCU,  
BGR555ToYCbCr411LS\_MCU  
BGR565ToYCbCr420,  
BGR555ToYCbCr420  
BGR565ToYCbCr422  
BGR555ToYCbCr422  
BGR565ToYCbCr422LS\_MCU,

BGR555ToYCbCr422LS\_MCU  
BGR565ToYCbCr444LS\_MCU,  
BGR555ToYCbCr444LS\_MCU  
BGR565ToYCrCb420  
BGR555ToYCrCb420  
BGR565ToYUV420  
BGR555ToYUV420  
BGRTToCbYCr422  
BGRTToHLS  
BGRTToLab  
BGRTToY\_JPEG  
BGRTToYCbCr\_JPEG  
BGRTToYCbCr411  
BGRTToYCbCr411LS\_MCU  
BGRTToYCbCr420  
BGRTToYCbCr422  
BGRTToYCbCr422LS\_MCU  
BGRTToYCbCr444LS\_MCU  
BGRTToYCoCg  
BGRTToYCoCg\_Rev  
BGRTToYCrCb420  
BiDirWeightBlock\_H264  
BiDirWeightBlockImplicit\_H264  
BlockMatch\_Integer\_16x16\_MVFAST  
BlockMatch\_Integer\_16x16\_SEA

### C

CalcGlobalMV\_MPEG4  
Canny  
CannyGetSize  
CbYCr422ToBGR  
CbYCr422ToRGB  
CbYCr422ToYCbCr411  
CbYCr422ToYCbCr420  
CbYCr422ToYCbCr420\_Rotate  
CbYCr422ToYCbCr422  
CbYCr422ToYCrCb420  
ChangeSpriteBrightness\_MPEG4  
CMYKToYCKK\_JPEG  
CMYKToYCKK411LS\_MCU  
CMYKToYCKK422LS\_MCU  
CMYKToYCKK444LS\_MCU

ColorToGray	DCT8x8Fwd
ColorTwist	DCT8x8FwdLS
ColorTwist32f	DCT8x8Inv
Compare	DCT8x8Inv_2x2,
CompareC	DCT8x8Inv_4x4
CompareEqualEps	DCT8x8Inv_AANTransposed_16s_C1R
CompareEqualEpsC	DCT8x8Inv_AANTransposed_16s_P2C2R
Complement	DCT8x8Inv_AANTransposed_16s8u_C1R
ComputeTextureErrorBlock	DCT8x8Inv_AANTransposed_16s8u_P2C2R
ComputeTextureErrorBlock_SAD	DCT8x8InvLSClip
Convert	DCTFwd
ConvFull	DCTFwdFree
ConvValid	DCTFwdGetBufSize
Copy	DCTFwdInitAlloc
Copy8x4HP,	DCTInv
Copy8x8HP,	DCTInvFree
Copy16x8HP,	DCTInvGetBufSize
Copy16x16HP	DCTInvInitAlloc
Copy8x8,	DCTQuantFwd8x8_JPEG
Copy16x16	DCTQuantFwd8x8LS_JPEG
Copy8x8QP_MPEG4,	DCTQuantInv8x8_JPEG
Copy16x8QP_MPEG4,	DCTQuantInv8x8LS_JPEG
Copy16x16QP_MPEG4	DecodeBlockCoef_Inter_H263
CopyConstBorder	DecodeBlockCoef_Inter_MPEG4
CopyReplicateBorder	DecodeBlockCoef_Intra_H263
CopySubpix	DecodeBlockCoef_Intra_MPEG4
CopySubpixIntersect	DecodeBlockCoef_IntraDOnly_MPEG4
CopyWrapBorder	DecodeCAEInterH_MPEG4,
CountInRange	DecodeCAEInterV_MPEG4
CountZeros8x8	DecodeCAEIntraH_MPEG4,
CreateMapCameraUndistort	DecodeCAEIntraV_MPEG4
CreateRLEncodeTable	DecodeCAVLCChromaDcCoeffs_H264
CrossCorrFull_Norm	DecodeCAVLCCoeffs_H264
CrossCorrFull_NormLevel	DecodeCBProgrAttach_JPEG2K
CrossCorrSame_Norm	DecodeCBProgrFree_JPEG2K
CrossCorrSame_NormLevel	DecodeCBProgrGetCurBitPlane_JPEG2K
CrossCorrValid_Norm	DecodeCBProgrGetPassCounter_JPEG2K
CrossCorrValid_NormLevel	DecodeCBProgrGetStateSize_JPEG2K
<b>D</b>	DecodeCBProgrInit_JPEG2K
DCT2x4x8Frw	DecodeCBProgrInitAlloc_JPEG2K
DCT2x4x8Inv	DecodeCBProgrSetPassCounter_JPEG2K
	DecodeCBProgrStep_JPEG2K

DecodeCodeBlock\_JPEG2K  
DecodeCoeffsInter\_H261  
DecodeCoeffsInter\_H263  
DecodeCoeffsInter\_MPEG4  
DecodeCoeffsInterRVLCBack\_MPEG4  
DecodeCoeffsIntra\_H261  
DecodeCoeffsIntra\_H263  
DecodeCoeffsIntra\_MPEG4  
DecodeCoeffsIntraRVLCBack\_MPEG4  
DecodeDCIntra\_H263  
DecodeDCIntra\_MPEG4  
DecodeExpGolombOne\_H264  
DecodeGetBufSize\_JPEG2K  
DecodeHuffman8x8\_ACFirst\_JPEG  
DecodeHuffman8x8\_ACFine\_JPEG  
DecodeHuffman8x8\_DCFirst\_JPEG  
DecodeHuffman8x8\_DCFine\_JPEG  
DecodeHuffman8x8\_Direct\_JPEG  
DecodeHuffman8x8\_JPEG  
DecodeHuffmanOne  
DecodeHuffmanOne\_JPEG  
DecodeHuffmanPair  
DecodeHuffmanSpecFree\_JPEG  
DecodeHuffmanSpecGetBufSize\_JPEG  
DecodeHuffmanSpecInit\_JPEG  
DecodeHuffmanSpecInitAlloc\_JPEG  
DecodeHuffmanStateFree\_JPEG  
DecodeHuffmanStateGetBufSize\_JPEG  
DecodeHuffmanStateInit\_JPEG  
DecodeHuffmanStateInitAlloc\_JPEG  
DecodeMV\_BVOP\_Backward\_MPEG4  
DecodeMV\_BVOP\_Direct\_MPEG4  
DecodeMV\_BVOP\_DirectSkip\_MPEG4  
DecodeMV\_BVOP\_Forward\_MPEG4  
DecodeMV\_BVOP\_Interpolate\_MPEG4  
DecodeMVS\_MPEG4  
DecodePadMV\_PVOP\_MPEG4  
DecodeVLCZigzag\_Inter\_MPEG4  
DecodeVLCZigzag\_IntraDCVLC\_MPEG4,  
DecodeVLCZigzag\_IntraACVLC\_MPEG4  
DeconvFFT  
DeconvFFTFree

DeconvFFTInitAlloc  
DeconvLR  
DeconvLRFree  
DeconvLRInitAlloc  
DeinterlaceFilterTriangle  
DequantTransformResidual\_H264  
DequantTransformResidual\_SISP\_H264  
DequantTransformResidualAndAdd\_H264  
DFTFree  
DFTFwd  
DFTGetBufSize  
DFTInitAlloc  
DFTInv  
DiffPredFirstRow\_JPEG  
DiffPredRow\_JPEG  
Dilate  
Dilate3x3  
DilateBorderReplicate  
DistanceTransform  
Div  
DivC  
DownsampleFour\_H263  
Dup

## E

EdgesDetect16x16  
EigenValsVecs  
EigenValsVecsGetBufferSize  
EncodeChromaDcCoeffsCAVLC\_H264  
EncodeCoeffsCAVLC\_H264  
EncodeCoeffsInter\_H261  
EncodeCoeffsInter\_H263  
EncodeCoeffsInter\_MPEG4  
EncodeCoeffsIntra\_H261  
EncodeCoeffsIntra\_H263  
EncodeCoeffsIntra\_MPEG4  
EncodeDCIntra\_H263  
EncodeDCIntra\_MPEG4  
EncodeFree\_JPEG2K  
EncodeGetDist\_JPEG2K  
EncodeGetRate\_JPEG2K  
EncodeGetTermPassLen\_JPEG2K

---

EncodeHuffman8x8_ACFirst_JPEG	FilterColumn
EncodeHuffman8x8_ACRrefine_JPEG	FilterColumn32f
EncodeHuffman8x8_DCFirst_JPEG	FilterColumnPipeline
EncodeHuffman8x8_DCRrefine_JPEG	FilterColumnPipeline_Low
EncodeHuffman8x8_Direct_JPEG	FilterColumnPipelineGetBufferSize
EncodeHuffman8x8_JPEG	FilterColumnPipelineGetBufferSize_Low
EncodeHuffmanOne_JPEG	FilterDeblocking16x16HorEdge_H263,
EncodeHuffmanRawTableInit_JPEG	FilterDeblocking16x16VerEdge_H263
EncodeHuffmanSpecFree_JPEG	FilterDeblocking8x8HorEdge_H263,
EncodeHuffmanSpecGetBufSize_JPEG	FilterDeblocking8x8VerEdge_H263
EncodeHuffmanSpecInit_JPEG	FilterDeblocking8x8HorEdge_MPEG4,
EncodeHuffmanSpecInitAlloc_JPEG	FilterDeblocking8x8VerEdge_MPEG4
EncodeHuffmanStateFree_JPEG	FilterDeblockingChroma_HorEdge_H264
EncodeHuffmanStateGetBufSize_JPEG	FilterDeblockingChroma_VerEdge_H264
EncodeHuffmanStateInit_JPEG	FilterDeblockingChroma_VerEdge_MBAFF_
EncodeHuffmanStateInitAlloc_JPEG	H264
EncodeInitAlloc_JPEG2K	FilterDeblockingLuma_HorEdge_H264
EncodeLoadCodeBlock_JPEG2K	FilterDeblockingLuma_VerEdge__MBAFF_H
EncodeMV_MPEG4	264
EncodeStoreBits_JPEG2K	FilterDeblockingLuma_VerEdge_H264
EncodeVLCZigzag_Inter_MPEG4	FilterDeringingSmooth8x8_MPEG4
EncodeVLCZigzag_IntraDCVLC_MPEG4,	FilterDeringingThreshold_MPEG4
EncodeVLCZigzag_IntraACVLC_MPEG4	FilterGauss
Erode	FilterHipass
Erode3x3	FilterLaplace
ErodeBorderReplicate	FilterLaplacianBorder
Exp	FilterLaplacianGetBufferSize
ExpandFrame_H263	FilterLowpass
ExpandPlane_H264	FilterLowpassBorder
<b>F</b>	FilterLowpassGetBufferSize
FFTFree	FilterMax
FFTFwd	FilterMaxBorderReplicate
FFTGetBufSize	FilterMaxGetBufferSize
FFTInitAlloc	FilterMedian
FFTInv	FilterMedianColor
Filter	FilterMedianCross
Filter32f	FilterMedianHoriz
Filter8x8_H261	FilterMedianVert
FilterBlockBoundaryHorEdge_H263,	FilterMin
FilterBlockBoundaryVerEdge_H263	FilterMinBorderReplicate
FilterBox	FilterMinGetBufferSize
	FilterPrewittHoriz



FilterPrewittVert	FloodFillGetSize_Grad
FilterRobertsDown	FrameFieldSAD16x16
FilterRobertsUp	Free
FilterRow	FreeHuffmanTable_DV
FilterRow32f	<b>G</b>
FilterRowBorderPipeline	GammaFwd
FilterRowBorderPipeline_Low	GammaInv
FilterRowBorderPipelineGetBufferSize	GenScaleLevel8x8_H264
FilterRowBorderPipelineGetBufferSize_Low	GenSobelKernel
FilterScharrHoriz	GetAffineBound
FilterScharrHorizBorder	GetAffineQuad
FilterScharrHorizGetBufferSize	GetAffineTransform
FilterScharrVert	GetBilinearBound
FilterScharrVertBorder	GetBilinearQuad
FilterScharrVertGetBufferSize	GetBilinearTransform
FilterSharpen	GetCentralMoment
FilterSobelCross	GetDiff16x16
FilterSobelCrossBorder	GetDiff16x16B
FilterSobelCrossGetBufferSize	GetDiff16x8
FilterSobelHoriz, FilterSobelHoriz-Mask	GetDiff16x8B
FilterSobelHorizBorder	GetDiff4x4
FilterSobelHorizGetBufferSize	GetDiff8x16
FilterSobelHorizSecond	GetDiff8x16B
FilterSobelHorizSecondBorder	GetDiff8x4
FilterSobelHorizSecondGetBufferSize	GetDiff8x4B
FilterSobelVert, FilterSobelVertMask	GetDiff8x8
FilterSobelVertBorder,	GetDiff8x8B
FilterSobelNegVertBorder	GetDistanceTransformMask
FilterSobelVertGetBufferSize,	GetHaarClassifierSize
FilterSobelNegVertGetBufferSize	GetHuffmanStatistics8x8_ACFfirst_JPEG
FilterSobelVertSecond	GetHuffmanStatistics8x8_ACFrefine_JPEG
FilterSobelVertSecondBorder	GetHuffmanStatistics8x8_DCFfirst_JPEG
FilterSobelVertSecondGetBufferSize	GetHuffmanStatistics8x8_JPEG
FilterWiener	GetHuffmanStatisticsOne_JPEG
FilterWienerGetBufferSize	GetHuMoments
FindMVpred_MPEG4	GetLibVersion
FloodFill	GetNormalizedCentralMoment
FloodFill_Grad	GetNormalizedSpatialMoment
FloodFill_Range	GetPerspectiveBound
FloodFillGetSize	GetPerspectiveQuad
	GetPerspectiveTransform

GetPyramidDownROI

GetPyramidUpROI

GetResizeFract

GetRotateBound

GetRotateQuad

GetRotateShift

GetShearBound

GetShearQuad

GetSpatialMoment

GradientColorToGray

GrayErodeBorder

## H

HaarClassifierFree

HaarClassifierInitAlloc

HistogramEven

HistogramRange

HLSToBGR

HLSToRGB

HSVToRGB

HuffmanDecodeSegment\_DV

HuffmanRunLevelTableInitAlloc

HuffmanTableFree

HuffmanTableInitAlloc

## I

ICTFwd\_JPEG2K

ICTInv\_JPEG2K

ImageJaehne

ImageRamp

InitAllocHuffmanTable\_DV

Integral

InterpolateAverage8x4,

InterpolateAverage8x8,

InterpolateAverage16x8,

InterpolateAverage16x16

InterpolateBlock\_H264

InterpolateChroma\_H264

InterpolateChromaBottom\_H264

InterpolateChromaTop\_H264

InterpolateLuma\_H264

InterpolateLumaBottom\_H264

InterpolateLumaTop\_H264

ippGetStatusString

ippjGetLibVersion

## J

Join422LS\_MCU

## L

LabToBGR

LimitMVToRect\_MPEG4

Ln

LShiftC

LUT

LUT\_Cubic

LUT\_Linear

LUTPalette

LUVToRGB

## M

Magnitude

MagnitudePack

Malloc

Max

MaxIndx

MC16x16

MC16x16B

MC16x4

MC16x4B

MC16x8

MC16x8B

MC16x8UV

MC16x8UVB

MC2x2

MC2x2B

MC2x4

MC2x4B

MC4x2

MC4x2B

MC4x4

MC4x4B

MC4x8

MC4x8B

MC8x16

MC8x16B  
MC8x4  
MC8x4B  
MC8x8  
MC8x8B  
Mean  
Mean\_StdDev  
MeanAbsDev16x16  
Min  
MinEigenVal  
MinEigenValGetBufferSize  
MinIndx  
MinMax  
MinMaxIndx  
Mirror  
MomentFree  
MomentGetStateSize  
MomentInit  
MomentInitAlloc  
Moments  
MorphAdvFree  
MorphAdvGetSize  
MorphAdvInit  
MorphAdvInitAlloc  
MorphBlackhatBorder  
MorphCloseBorder  
MorphDilateBorder  
MorphGradientBorder  
MorphGrayFree  
MorphGrayGetSize  
MorphGrayInit  
MorphGrayInitAlloc  
MorphologyFree  
MorphologyGetSize  
MorphologyInit  
MorphologyInitAlloc  
MorphOpenBorder  
MorphReconstructDilate  
MorphReconstructErode  
MorphReconstructGetBufferSize  
MorphTophatBorder  
MotionEstimation\_16x16\_MVFAST

MotionEstimation\_16x16\_SEA  
Mul  
MulC  
MulCScale  
MulPack  
MulPackConj  
MulScale

## N

Norm\_Inf  
Norm\_L1  
Norm\_L2  
NormDiff\_Inf  
NormDiff\_L1  
NormDiff\_L2  
NormRel\_Inf  
NormRel\_L1  
NormRel\_L2  
Not

## O

OBMC8x8HP\_MPEG4,  
OBMC16x16HP\_MPEG4,  
OBMC8x8QP\_MPEG4  
OpticalFlowPyrLK  
OpticalFlowPyrLKFree  
OpticalFlowPyrLKInitAlloc  
Or  
OrC

## P

PackToCplxExtend  
PadCurrent\_16x16\_MPEG4,  
PadCurrent\_8x8\_MPEG4  
PadMBGray\_MPEG4  
PadMBHorizontal\_MPEG4  
PadMBOpaque\_MPEG4  
PadMBPartial\_MPEG4  
PadMBTransparent\_MPEG4  
PadMBVertical\_MPEG4  
PadMV\_MPEG4  
Phase  
PhasePack

PolarToCart  
 PredictIntra\_16x16\_H264  
 PredictIntra\_4x4\_H264  
 PredictIntraChroma8x8\_H264  
 PredictReconCoefIntra\_MPEG4  
 PutIntraBlock  
 PutNonIntraBlock  
 PyramidFree  
 PyramidInitAlloc  
 PyramidLayerDown  
 PyramidLayerDownFree  
 PyramidLayerDownInitAlloc  
 PyramidLayerUp  
 PyramidLayerUpFree  
 PyramidLayerUpInitAlloc  
 PyrDown  
 PyrDownGetBufSize  
 PyrUp  
 PyrUpGetBufSize

## Q

QualityIndex  
 Quant\_MPEG2  
 QuantFwd8x8\_JPEG  
 QuantFwdRawTableInit\_JPEG  
 QuantFwdTableInit\_JPEG  
 QuantInter\_H263  
 QuantIntra\_H263  
 QuantIntra\_MPEG2  
 QuantIntra\_MPEG4,  
 QuantInter\_MPEG4  
 QuantIntraGetSize\_MPEG4,  
 QuantInterGetSize\_MPEG4  
 QuantIntraInit\_MPEG4,  
 QuantInterInit\_MPEG4  
 QuantInv\_DV  
 QuantInv\_MPEG2  
 QuantInv8x8\_JPEG  
 QuantInvInter\_H263  
 QuantInvIntra\_H263  
 QuantInvIntra\_MPEG2  
 QuantInvIntra\_MPEG4,

QuantInvInter\_MPEG4  
 QuantInvIntraGetSize\_MPEG4,  
 QuantInvInterGetSize\_MPEG4  
 QuantInvIntraInit\_MPEG4,  
 QuantInvInterInit\_MPEG4  
 QuantInvTableInit\_JPEG  
 QuantLuma8x8\_H264  
 QuantLuma8x8Inv\_H264

## R

RCTFwd\_JPEG2K  
 RCTInv\_JPEG2K  
 ReconstructChromaInterMB\_H264  
 ReconstructChromaIntra4x4MB\_H264  
 ReconstructChromaIntraHalves4x4MB\_H264  
 ReconstructChromaIntraHalvesMB\_H264  
 ReconstructChromaIntraMB\_H264  
 ReconstructCoeffsInter\_H261  
 ReconstructCoeffsInter\_H263  
 ReconstructCoeffsInter\_MPEG4  
 ReconstructCoeffsIntra\_H261  
 ReconstructCoeffsIntra\_H263  
 ReconstructDCTBlock\_MPEG1  
 ReconstructDCTBlock\_MPEG2  
 ReconstructDCTBlockIntra\_MPEG1  
 ReconstructDCTBlockIntra\_MPEG2  
 ReconstructLumaInter4x4MB\_H264  
 ReconstructLumaInter8x8MB\_H264  
 ReconstructLumaInterMB\_H264  
 ReconstructLumaIntra\_16x16MB\_H264  
 ReconstructLumaIntra16x16MB\_H264  
 ReconstructLumaIntra4x4MB\_H264  
 ReconstructLumaIntra8x8MB\_H264  
 ReconstructLumaIntraHalf4x4MB\_H264  
 ReconstructLumaIntraHalf8x8MB\_H264  
 ReconstructLumaIntraHalfMB\_H264  
 ReconstructLumaIntraMB\_H264  
 ReconstructPredFirstRow\_JPEG  
 ReconstructPredRow\_JPEG  
 RectStdDev  
 ReduceBits

Remap	SampleDown422LS_MCU
Resample_H263	SampleDown444LS_MCU
Resize	SampleDownH2V1_JPEG
ResizeCCRotate	SampleDownH2V2_JPEG
ResizeCenter	SampleDownRowH2V1_Box_JPEG
ResizeShift	SampleDownRowH2V2_Box_JPEG
ResizeSqrPixel	SampleLine
ResizeSqrPixelGetBufSize	SampleUp411LS_MCU
ResizeYUV422	SampleUp422LS_MCU
RGB565ToYCbCr_JPEG,	SampleUp444LS_MCU
RGB555ToYCbCr_JPEG	SampleUpH2V1_JPEG
RGBToCbYCr422	SampleUpH2V2_JPEG
RGBToCbYCr422Gamma	SampleUpRowH2V1_Triangle_JPEG
RGBToGray	SampleUpRowH2V2_Triangle_JPEG
RGBToHLS	SBGRToYCoCg
RGBToHSV	SBGRToYCoCg_Rev
RGBToLUV	Scale
RGBToXYZ	ScanFwd
RGBToY_JPEG	ScanInv
RGBToYCbCr	Set
RGBToYCbCr_JPEG	Shear
RGBToYCbCr411LS_MCU	SpatialInterpolation_H263
RGBToYCbCr420	Split422LS_MCU
RGBToYCbCr422	Sqr
RGBToYCbCr422LS_MCU	SqrDiff16x16
RGBToYCbCr444LS_MCU	SqrDiff16x16B
RGBToYCC	SqrDistanceFull_Norm
RGBToYUV	SqrDistanceSame_Norm
RGBToYUV420	SqrDistanceValid_Norm
RGBToYUV422	SqrIntegral
Rotate	Sqrt
RotateCenter	SSD4x4
RShiftC	SSD8x8
<b>S</b>	Sub
SAD16x16	Sub128_JPEG
SAD16x16Blocks4x4	Sub8x8,
SAD16x16Blocks8x8	Sub16x16
SAD16x8	SubC
SAD4x4	SubSAD8x8
SAD8x8	Sum
SampleDown411LS_MCU	SumNorm_VOP_MPEG4
	SumsDiff16x16Blocks4x4

SumsDiff8x8Blocks4x4  
 SumWindowColumn  
 SumWindowRow  
 SwapChannels

## T

Threshold  
 Threshold\_GT  
 Threshold\_GTVal  
 Threshold\_LT  
 Threshold\_LTVal  
 Threshold\_LTValGTVal  
 Threshold\_Val  
 TiltedHaarClassifierInitAlloc  
 TiltedIntegral  
 TiltedRectStdDev  
 TiltedSqrIntegral  
 TiltHaarFeatures  
 TransformDequantChromaDC\_H264  
 TransformDequantChromaDC\_SISP\_H264  
 TransformDequantLumaDC\_H264  
 TransformLuma8x8Fwd\_H264  
 TransformLuma8x8InvAddPred\_H264  
 TransformPrediction\_H264  
 TransformQuantChromaDC\_H264  
 TransformQuantLumaDC\_H264  
 TransformQuantResidual\_H264  
 Transpose  
 TransRecBlockCoef\_inter\_MPEG4  
 TransRecBlockCoef\_intra\_MPEG4

## U

UndistortGetSize  
 UndistortRadial  
 UniDirWeightBlock\_H264  
 UpdateMotionHistory  
 UpdateQP\_MPEG4  
 UpdateRCModel\_MPEG4  
 UpsampleFour\_H263  
 UpsampleFour8x8\_H263

## V

Variance16x16

VarMean8x8  
 VarMeanDiff16x16  
 VarMeanDiff16x8

## W

WarpAffine  
 WarpAffineBack  
 WarpAffineQuad  
 WarpBilinear  
 WarpBilinearBack  
 WarpBilinearQuad  
 WarpChroma\_MPEG4  
 WarpGetSize\_MPEG4  
 WarpInit\_MPEG4  
 WarpLuma\_MPEG4  
 WarpPerspective  
 WarpPerspectiveBack  
 WarpPerspectiveQuad  
 WeightedAverage\_H264  
 WinBartlett,  
 WinBartlettSep  
 WinHamming,  
 WinHammingSep  
 WTFwd  
 WTFwd\_B53\_JPEG2K  
 WTFwd\_D97\_JPEG2K  
 WTFwdCol\_B53\_JPEG2K  
 WTFwdCol\_D97\_JPEG2K  
 WTFwdColLift\_B53\_JPEG2K  
 WTFwdColLift\_D97\_JPEG2K  
 WTFwdFree  
 WTFwdGetBufSize  
 WTFwdInitAlloc  
 WTFwdRow\_B53\_JPEG2K  
 WTFwdRow\_D97\_JPEG2K  
 WTGetBufSize\_B53\_JPEG2K  
 WTGetBufSize\_D97\_JPEG2K  
 WTInv  
 WTInv\_B53\_JPEG2K  
 WTInv\_D97\_JPEG2K  
 WTInvCol\_B53\_JPEG2K  
 WTInvCol\_D97\_JPEG2K

WTInvColLift\_B53\_JPEG2K  
WTInvColLift\_D97\_JPEG2K  
WTInvFree  
WTInvGetBufSize  
WTInvInitAlloc  
WTInvRow\_B53\_JPEG2K  
WTInvRow\_D97\_JPEG2K

## X

Xor  
XorC  
XYZToRGB

## Y

YCbC422ToBGR565Dither  
YCbC422ToBGR555Dither  
YCbCr422ToBGR444Dither  
YCbCr411  
YCbCr411ToBGR  
YCbCr411ToBGR565,  
YCbCr411ToBGR555  
YCbCr411ToBGR565LS\_MCU,  
YCbCr411ToBGR555LS\_MCU  
YCbCr411ToBGR5LS\_MCU  
YCbCr411ToBGR5LS\_MC  
YCbCr411ToYCbCr420  
YCbCr411ToYCbCr422  
YCbCr411ToYCrCb420  
YCbCr411ToYCrCb422  
YCbCr420  
YCbCr420ToBGR  
YCbCr420ToBGR565  
YCbCr420ToBGR555  
YCbCr420ToBGR444  
YCbCr420ToBGR565\_Rotate  
YCbCr420ToBGR565Dither  
YCbCr420ToBGR555Dither  
YCbCr420ToBGR444Dither  
YCbCr420ToCbYCr422  
YCbCr420ToRGB  
YCbCr420ToRGB565  
YCbCr420ToRGB555

YCbCr420ToRGB444  
YCbCr420ToRGB565Dither  
YCbCr420ToRGB555Dither  
YCbCr420ToRGB444Dither  
YCbCr420ToYCbCr411  
YCbCr420ToYCbCr422  
YCbCr420ToYCbCr422\_Filter  
YCbCr420ToYCrCb420  
YCbCr420ToYCrCb420\_Filter  
YCbCr422  
YCbCr422ToBGR  
YCbCr422ToBGR565  
YCbCr422ToBGR555  
YCbCr422ToBGR444  
YCbCr422ToBGR565LS\_MCU,  
YCbCr422ToBGR555LS\_MCU  
YCbCr422ToBGR5LS\_MCU  
YCbCr422ToCbYCr422  
YCbCr422ToRGB  
YCbCr422ToRGB565  
YCbCr422ToRGB555  
YCbCr422ToRGB444  
YCbCr422ToRGB565Dither  
YCbCr422ToRGB555Dither  
YCbCr422ToRGB444Dither  
YCbCr422ToRGB5LS\_MCU  
YCbCr422ToYCbCr411  
YCbCr422ToYCbCr420  
YCbCr422ToYCrCb420  
YCbCr422ToYCrCb422  
YCbCr444ToBGR565LS\_MCU,  
YCbCr444ToBGR555LS\_MCU  
YCbCr444ToBGR5LS\_MCU  
YCbCr444ToBGR5LS\_MC  
YCbCrToBGR  
YCbCrToBGR\_JPEG  
YCbCrToBGR565  
YCbCrToBGR555  
YCbCrToBGR444  
YCbCrToBGR565\_JPEG,  
YCbCrToBGR555\_JPEG  
YCbCrToBGR565Dither

---

YCbCrToBGR555Dither  
 YCbCrToBGR444Dither  
 YCbCrToRGB  
 YCbCrToRGB\_JPEG  
 YCbCrToRGB565\_JPEG,  
 YCbCrToRGB555\_JPEG  
 YCbCrToRGB565Dither  
 YCbCrToRGB555Dither  
 YCbCrToRGB444Dither  
 YCbCrToRGB565  
 YCbCrToRGB555  
 YCbCrToRGB444  
 YCCK411ToCMYKLS\_MCU  
 YCCK422ToCMYKLS\_MCU  
 YCCK444ToCMYKLS\_MCU  
 YCCKToCMYK\_JPEG  
 YCCToRGB  
 YCoCgToBGR  
 YCoCgToBGR\_Rev  
 YCoCgToSBGR  
 YCoCgToSBGR\_Rev  
 YCrCb411ToYCbCr422\_5MBDV,  
 YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV,  
 YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV,  
 YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV  
 YCrCb411ToYCbCr422\_EdgeDV,  
 YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV,  
 YCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV,  
 YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV  
 YCrCb420ToCbYCr422  
 YCrCb420ToYCbCr411  
 YCrCb420ToYCbCr420  
 YCrCb420ToYCbCr422  
 YCrCb420ToYCbCr422\_5MBDV,  
 YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV,  
 YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV,  
 YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV  
 YCrCb420ToYCbCr422\_Filter  
 YCrCb422ToYCbCr411  
 YCrCb422ToYCbCr420  
 YCrCb422ToYCbCr422  
 YCrCb422ToYCbCr422\_5MBDV,  
 YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV,  
 YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV,  
 YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV  
 YUV420ToBGR  
 YUV420ToBGR565  
 YUV420ToBGR555  
 YUV420ToBGR444  
 YUV420ToBGR565Dither  
 YUV420ToBGR555Dither  
 YUV420ToBGR444Dither  
 YUV420ToRGB  
 YUV420ToRGB565  
 YUV420ToRGB555  
 YUV420ToRGB444  
 YUV420ToRGB565Dither  
 YUV420ToRGB555Dither  
 YUV420ToRGB444Dither  
 YUV422ToRGB  
 YUVToRGB

## Z

ZigzagFwd8x8  
 ZigzagInv8x8  
 ZigzagInvClassical\_Compact,  
 ZigzagInvHorizontal\_Compact,  
 ZigzagInvVertical\_Compact





# Support Functions

## Version Information Function

### GetLibVersion

Returns information about the used version of Intel IPP software for image processing.

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

## Status Information Function

### ippGetStatusString

Translates a status code into a message.

```
const char* ippGetStatusString(IppStatus StsCode);
```

## Memory Allocation Functions

### Malloc

Allocates memory aligned to 32-byte boundary.

```
Ipp8u* ippiMalloc_8u_C1(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp8u* ippiMalloc_8u_C2(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp8u* ippiMalloc_8u_C3(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp8u* ippiMalloc_8u_C4(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp8u* ippiMalloc_8u_AC4(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16u* ippiMalloc_16u_C1(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16u* ippiMalloc_16u_C2(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16u* ippiMalloc_16u_C3(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16u* ippiMalloc_16u_C4(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16u* ippiMalloc_16u_AC4(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16s* ippiMalloc_16s_C1(int widthPixels, int heightPixels, int*  
    pStepBytes);  
Ipp16s* ippiMalloc_16s_C2(int widthPixels, int heightPixels, int*  
    pStepBytes);
```

```
Ipp16s* ippiMalloc_16s_C3(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp16s* ippiMalloc_16s_C4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp16s* ippiMalloc_16s_AC4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32s* ippiMalloc_32s_C1(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32s* ippiMalloc_32s_C2(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32s* ippiMalloc_32s_C3(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32s* ippiMalloc_32s_C4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32s* ippiMalloc_32s_AC4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32f* ippiMalloc_32f_C1(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32f* ippiMalloc_32f_C2(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32f* ippiMalloc_32f_C3(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32f* ippiMalloc_32f_C4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32f* ippiMalloc_32f_AC4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32sc* ippiMalloc_32sc_C1(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32sc* ippiMalloc_32sc_C2(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32sc* ippiMalloc_32sc_C3(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32sc* ippiMalloc_32sc_C4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32sc* ippiMalloc_32sc_AC4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32fc* ippiMalloc_32fc_C1(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32fc* ippiMalloc_32fc_C2(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32fc* ippiMalloc_32fc_C3(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32fc* ippiMalloc_32fc_C4(int widthPixels, int heightPixels, int*
    pStepBytes);
Ipp32fc* ippiMalloc_32fc_AC4(int widthPixels, int heightPixels, int*
    pStepBytes);
```

**Free**

Frees memory allocated by the function `ippiMalloc`.

```
void ippiFree(void* ptr);
```

## Image Data Exchange and Initialization Functions

**Convert**

Converts pixel values of an image from one bit depth to another.

**Case 1: Conversion to increased bit depth**

```
IppStatus ippiConvert_1u8u_C1R(const Ipp8u* pSrc, int srcStep, int
    srcBitOffset, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_C1R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_C4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C3R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C1R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C3R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C4R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_AC4R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiConvert_16u32s_C1R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_C3R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_C4R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C3R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C1R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C3R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C4R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_AC4R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C1R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C3R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C4R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C1R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C3R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C4R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion to reduced bit depth:****integer to integer type**

```
IppStatus ippiConvert_1u8u_C1R(const Ipp8u* pSrc, int srcStep, int
    srcBitOffset, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiConvert_8u16u_C1R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_C4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C3R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_C4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u16s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C1R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C3R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_C4R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32s_AC4R(const Ipp8s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_C1R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_C3R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_C4R(const Ipp16u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32s_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C3R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8u32f_C4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiConvert_8u32f_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C1R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C3R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_C4R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_8s32f_AC4R(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C1R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C3R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_C4R(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16u32f_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C1R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C3R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_C4R(const Ipp16s* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiConvert_16s32f_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

### **floating point to integer type**

```
IppStatus ippiConvert_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8u_C3R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8u_C4R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8u_AC4R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8s_C1R(const Ipp32f* pSrc, int srcStep, Ipp8s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8s_C3R(const Ipp32f* pSrc, int srcStep, Ipp8s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8s_C4R(const Ipp32f* pSrc, int srcStep, Ipp8s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f8s_AC4R(const Ipp32f* pSrc, int srcStep, Ipp8s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

```
IppStatus ippiConvert_32f16u_C1R(const Ipp32f* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16u_C3R(const Ipp32f* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16u_C4R(const Ipp32f* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16u_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16s_C1R(const Ipp32f* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16s_C3R(const Ipp32f* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16s_C4R(const Ipp32f* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
IppStatus ippiConvert_32f16s_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

## Scale

Scales pixel values of a buffer and converts them to another bit depth.

### Case 1: Scaling with conversion to integer data of increased bit depth

```
IppStatus ippiScale_8u16u_C1R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16u_C4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16s_C3R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16s_C4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u16s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiScale_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
```



**Case 2: Scaling with conversion to floating-point data**

```
IppStatus ippiScale_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_8u32f_C3R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_8u32f_C4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_8u32f_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 3: Scaling of integer data with conversion to reduced bit depth**

```
IppStatus ippiScale_16u8u_C1R(const Ipp16u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16u8u_C4R(const Ipp16u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16u8u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16s8u_C1R(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16s8u_C3R(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16s8u_C4R(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_16s8u_AC4R(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_32s8u_C1R(const Ipp32s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_32s8u_C3R(const Ipp32s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_32s8u_C4R(const Ipp32s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
IppStatus ippiScale_32s8u_AC4R(const Ipp32s* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

**Case 4: Scaling of floating-point data with conversion to integer data type**

```
IppStatus ippiScale_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_32f8u_C3R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_32f8u_C4R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiScale_32f8u_AC4R(const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

## Set

Sets an array of pixels to a value.

### Case 1: Setting one-channel data to a value

```

IppStatus ippiSet_8u_C1R(Ipp8u value, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSet_16s_C1R(Ipp16s value, Ipp16s* pDst, int dstStep,
    IppiSize roiSize);
IppStatus ippiSet_32s_C1R(Ipp32s value, Ipp32s* pDst, int dstStep,
    IppiSize roiSize);
IppStatus ippiSet_32f_C1R(Ipp32f value, Ipp32f* pDst, int dstStep,
    IppiSize roiSize);

```

### Case 2: Setting each color channel to a specified value

```

IppStatus ippiSet_8u_C3R(const Ipp8u value[3], Ipp8u* pDst, int dstStep,
    IppiSize roiSize);
IppStatus ippiSet_8u_AC4R(const Ipp8u value[3], Ipp8u* pDst, int dstStep,
    IppiSize roiSize);
IppStatus ippiSet_16s_C3R(const Ipp16s value[3], Ipp16s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_16s_AC4R(const Ipp16s value[3], Ipp16s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32s_C3R(const Ipp32s value[3], Ipp32s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32s_AC4R(const Ipp32s value[3], Ipp32s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32f_C3R(const Ipp32f value[3], Ipp32f* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32f_AC4R(const Ipp32f value[3], Ipp32f* pDst, int
    dstStep, IppiSize roiSize);

```

### Case 3: Setting color channels and alpha channel to specified values

```

IppStatus ippiSet_8u_C4R(const Ipp8u value[4], Ipp8u* pDst, int dstStep,
    IppiSize roiSize);
IppStatus ippiSet_16s_C4R(const Ipp16s value[4], Ipp16s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32s_C4R(const Ipp32s value[4], Ipp32s* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiSet_32f_C4R(const Ipp32f value[4], Ipp32f* pDst, int
    dstStep, IppiSize roiSize);

```

### Case 4: Setting masked one-channel data to a value

```

IppStatus ippiSet_8u_C1MR(Ipp8u value, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiSet_16s_C1MR(Ipp16s value, Ipp16s* pDst, int dstStep,
    IppiSize roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiSet_32s_C1MR(Ipp32s value, Ipp32s* pDst, int dstStep,
    IppiSize roiSize, const Ipp8u* pMask, int maskStep);

```

```
IppStatus ippiSet_32f_C1MR(Ipp32f value, Ipp32f* pDst, int dstStep,  
    IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

**Case 5: Setting color channels of masked multi-channel data to specified values**

```
IppStatus ippiSet_8u_C3MR(const Ipp8u value[3], Ipp8u* pDst, int dstStep,  
    IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_8u_AC4MR(const Ipp8u value[3], Ipp8u* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_16s_C3MR(const Ipp16s value[3], Ipp16s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_16s_AC4MR(const Ipp16s value[3], Ipp16s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32s_C3MR(const Ipp32s value[3], Ipp32s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32s_AC4MR(const Ipp32s value[3], Ipp32s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32f_C3MR(const Ipp32f value[3], Ipp32f* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32f_AC4MR(const Ipp32f value[3], Ipp32f* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

**Case 6: Setting all channels of masked multi-channel data to specified values**

```
IppStatus ippiSet_8u_C4MR(const Ipp8u value[4], Ipp8u* pDst, int dstStep,  
    IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_16s_C4MR(const Ipp16s value[4], Ipp16s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32s_C4MR(const Ipp32s value[4], Ipp32s* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);  
IppStatus ippiSet_32f_C4MR(const Ipp32f value[4], Ipp32f* pDst, int  
    dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

**Case 7: Setting selected channel of multi-channel data to a value**

```
IppStatus ippiSet_8u_C3CR(Ipp8u value, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize);  
IppStatus ippiSet_8u_C4CR(Ipp8u value, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize);  
IppStatus ippiSet_16s_C3CR(Ipp16s value, Ipp16s* pDst, int dstStep,  
    IppiSize roiSize);  
IppStatus ippiSet_16s_C4CR(Ipp16s value, Ipp16s* pDst, int dstStep,  
    IppiSize roiSize);  
IppStatus ippiSet_32s_C3CR(Ipp32s value, Ipp32s* pDst, int dstStep,  
    IppiSize roiSize);  
IppStatus ippiSet_32s_C4CR(Ipp32s value, Ipp32s* pDst, int dstStep,  
    IppiSize roiSize);  
IppStatus ippiSet_32f_C3CR(Ipp32f value, Ipp32f* pDst, int dstStep,  
    IppiSize roiSize);  
IppStatus ippiSet_32f_C4CR(Ipp32f value, Ipp32f* pDst, int dstStep,  
    IppiSize roiSize);
```

## Copy

Copies pixel values between two buffers.

### Case 1: Copying all pixels of all color channels

```

IppStatus ippiCopy_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C3AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_AC4C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C3AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_AC4C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C3R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C4R(const Ipp32s* pSrc, int srcStep, Ipp32s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_AC4R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C3AC4R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_AC4C3R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);

```

```
IppStatus ippiCopy_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C3AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_AC4C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Operation on masked pixels only**

```
IppStatus ippiCopy_8u_C1MR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiCopy_8u_C3MR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiCopy_8u_C4MR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiCopy_8u_AC4MR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
IppStatus ippiCopy_16s_C1MR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_16s_C3MR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_16s_C4MR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_16s_AC4MR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32s_C1MR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32s_C3MR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32s_C4MR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32s_AC4MR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32f_C1MR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
IppStatus ippiCopy_32f_C3MR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
```

```
IppStatus ippiCopy_32f_C4MR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);

IppStatus ippiCopy_32f_AC4MR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int
    maskStep);
```

**Case 3: Copying of a selected channel in a multi-channel image**

```
IppStatus ippiCopy_8u_C3CR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C4CR(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C3CR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C4CR(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C3CR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C4CR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C3CR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C4CR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 4: Copying of a selected channel to a one-channel image**

```
IppStatus ippiCopy_8u_C3C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C4C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C3C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C4C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C3C1R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C4C1R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C3C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C4C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 5: Copying a one-channel image to a multi-channel image**

```
IppStatus ippiCopy_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C1C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

```
IppStatus ippiCopy_16s_C1C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C1C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C1C3R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C1C4R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C1C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C1C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 6: Splitting color image into separate planes**

```
IppStatus ippiCopy_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* const
    pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_C3P3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    const pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_C3P3R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    const pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_C3P3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    const pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_C4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* const
    pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_P3C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    const pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_P3C3R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    const pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_P3C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    const pDst[4], int dstStep, IppiSize roiSize);
```

**Case 7: Composing color image from separate planes**

```
IppStatus ippiCopy_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_P3C3R(const Ipp16s* const pSrc[3], int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_P3C3R(const Ipp32s* const pSrc[3], int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32f_P3C3R(const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_8u_P4C4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_16s_P4C4R(const Ipp16s* const pSrc[4], int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCopy_32s_P4C4R(const Ipp32s* const pSrc[4], int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiCopy_32f_P4C4R(const Ipp32f* const pSrc[4], int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

## CopyConstBorder

Copies pixels values between two buffers and adds the border pixels with a constant value.

### Case 1: Operation on one-channel data

```
IppStatus ippiCopyConstBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, Ipp8u value);
IppStatus ippiCopyConstBorder_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, Ipp16s value);
IppStatus ippiCopyConstBorder_32s_C1R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, Ipp32s value);
```

### Case 2: Operation on multi-channel data

```
IppStatus ippiCopyConstBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp8u value[3]);
IppStatus ippiCopyConstBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp8u value[3]);
IppStatus ippiCopyConstBorder_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp16s value[3]);
IppStatus ippiCopyConstBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp16s value[3]);
IppStatus ippiCopyConstBorder_32s_C3R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp32s value[3]);
IppStatus ippiCopyConstBorder_32s_AC4R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp32s value[3]);
IppStatus ippiCopyConstBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp8u value[4]);
IppStatus ippiCopyConstBorder_16s_C4R(const Ipp16s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp16s value[4]);
IppStatus ippiCopyConstBorder_32s_C4R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth, const Ipp32s value[4]);
```



## CopyReplicateBorder

Copies pixel values between two buffers and adds the replicated border pixels.

### Case 1: Not-in-place operation

```
IppStatus ippiCopyReplicateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C1R(const Ipp16s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C3R(const Ipp16s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C4R(const Ipp16s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C1R(const Ipp32s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C3R(const Ipp32s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C4R(const Ipp32s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_AC4R(const Ipp32s* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize
    dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

### Case 2: Copying with conversion to integer data

```
IppStatus ippiCopyReplicateBorder_8u_C1IR(const Ipp8u* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_8u_C3IR(const Ipp8u* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
```

```
IppStatus ippiCopyReplicateBorder_8u_C4IR(const Ipp8u* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_8u_AC4IR(const Ipp8u* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C1IR(const Ipp16s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C3IR(const Ipp16s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_C4IR(const Ipp16s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_16s_AC4IR(const Ipp16s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C1IR(const Ipp32s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C3IR(const Ipp32s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_C4IR(const Ipp32s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyReplicateBorder_32s_AC4IR(const Ipp32s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
```

## CopyWrapBorder

Copies pixel values between two buffers and adds the border pixels.

```
IppStatus ippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize,
    int topBorderWidth, int leftBorderWidth);
IppStatus ippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc, int
    srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderWidth, int leftBorderWidth);
```

## CopySubpix

Copies pixel values between two buffers with subpixel precision.

**Case 1: Copying without conversion or with conversion to floating point data**

```
IppStatus ippiCopySubpix_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
IppStatus ippiCopySubpix_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
IppStatus ippiCopySubpix_16u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
IppStatus ippiCopySubpix_16u32f_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
IppStatus ippiCopySubpix_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
```

#### **Case 2: Copying with conversion to integer data**

```
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy,
    int scaleFactor);
```

## **CopySubpixIntersect**

Copies pixel values of the intersection with specified window with subpixel precision.

#### **Case 1: Copying without conversion or with conversion to floating point data**

```
IppStatus ippiCopySubpixIntersect_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax);
IppStatus ippiCopySubpixIntersect_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax);
IppStatus ippiCopySubpixIntersect_16u_C1R(const Ipp16u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax);
IppStatus ippiCopySubpixIntersect_16u32f_C1R(const Ipp16u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax);
IppStatus ippiCopySubpixIntersect_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax);
```

#### **Case 2: Copying with conversion to integer data**

```
IppStatus ippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint2D32f point, IppiPoint* pMin, IppiPoint* pMax,
    int scaleFactor);
```

## Dup

Copies gray scale image to all channels of the color image.

```
IppStatus, ippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize);
```

## Transpose

Transpose a source image.

### Case 1: Not-in-place operation

```
IppStatus ippiTranspose_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_32s_C3R(const Ipp32s* pSrc, int srcStep, Ipp32s*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiTranspose_32s_C4R(const Ipp32s* pSrc, int srcStep, Ipp32s*  
    pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiTranspose_8u_C1IR(const Ipp8u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);  
IppStatus ippiTranspose_8u_C3IR(const Ipp8u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);  
IppStatus ippiTranspose_8u_C4IR(const Ipp8u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);  
IppStatus ippiTranspose_16u_C1IR(const Ipp16u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);  
IppStatus ippiTranspose_16u_C3IR(const Ipp16u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

```
IppStatus ippiTranspose_16u_C4IR(const Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiTranspose_32s_C1IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiTranspose_32s_C3IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiTranspose_32s_C4IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

## SwapChannels

Changes the order of channels of the image.

### Case 1: Not-in-place operation

```
IppStatus ippiSwapChannels_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_16u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_32s_C3R(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_32s_AC4R(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

### Case 2: In-place operation

```
IppStatus ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const int dstOrder[3]);
```

## AddRandUniform\_Direct

Generates random samples with uniform distribution and adds them to an image data.

```
IppStatus ippiAddRandUniform_Direct_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u low, Ipp8u high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_8u_C3IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u low, Ipp8u high, unsigned int*
    pSeed);
```

```
IppStatus ippiAddRandUniform_Direct_8u_C4IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u low, Ipp8u high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_8u_AC4IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u low, Ipp8u high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_16s_C1IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s low, Ipp16s high, unsigned
    int* pSeed);
IppStatus ippiAddRandUniform_Direct_16s_C3IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s low, Ipp16s high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_16s_C4IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s low, Ipp16s high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_16s_AC4IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s low, Ipp16s high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_32f_C1IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f low, Ipp32f high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_32f_C3IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f low, Ipp32f high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_32f_C4IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f low, Ipp32f high, unsigned int*
    pSeed);
IppStatus ippiAddRandUniform_Direct_32f_AC4IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f low, Ipp32f high, unsigned int*
    pSeed);
```

## AddRandGauss\_Direct

Generates random samples with Gaussian distribution and adds them to an image data.

```
IppStatus ippiAddRandGauss_Direct_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u mean, Ipp8u stdev, unsigned int* pSeed);
IppStatus ippiAddRandGauss_Direct_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u mean, Ipp8u stdev, unsigned int* pSeed);
IppStatus ippiAddRandGauss_Direct_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u mean, Ipp8u stdev, unsigned int* pSeed);
IppStatus ippiAddRandGauss_Direct_8u_AC4IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u mean, Ipp8u stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_16s_C1IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s mean, Ipp16s stdev, unsigned int*
    pSeed);
```

```
IppStatus ippiAddRandGauss_Direct_16s_C3IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s mean, Ipp16s stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_16s_C4IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s mean, Ipp16s stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_16s_AC4IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s mean, Ipp16s stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_32f_C1IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f mean, Ipp32f stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_32f_C3IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f mean, Ipp32f stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_32f_C4IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f mean, Ipp32f stdev, unsigned int*
    pSeed);
IppStatus ippiAddRandGauss_Direct_32f_AC4IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f mean, Ipp32f stdev, unsigned int*
    pSeed);
```

## ImageJaehne

Creates Jaehne test image.

```
IppStatus ippiImageJaehne_8u_C1R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8u_C3R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8u_C4R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8u_AC4R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8s_C1R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8s_C3R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8s_C4R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_8s_AC4R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_16u_C1R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippiImageJaehne_16u_C3R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize);
```

```
IppStatus ippImageJaehne_16u_C4R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_16u_AC4R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_16s_C1R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_16s_C3R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_16s_C4R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_16s_AC4R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32s_C1R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32s_C3R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32s_C4R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32s_AC4R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32f_C1R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32f_C3R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32f_C4R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize);
IppStatus ippImageJaehne_32f_AC4R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize);
```

## ImageRamp

Creates a test image that has an intensity ramp.

```
IppStatus ippImageRamp_8u_C1R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8u_C3R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8u_C4R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8u_AC4R(Ipp8u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8s_C1R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8s_C3R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippImageRamp_8s_C4R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
```



```
IppStatus ippiImageRamp_8s_AC4R(Ipp8s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16u_C1R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16u_C3R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16u_C4R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16u_AC4R(Ipp16u* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16s_C1R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16s_C3R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16s_C4R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_16s_AC4R(Ipp16s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32s_C1R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32s_C3R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32s_C4R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32s_AC4R(Ipp32s* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32f_C1R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32f_C3R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32f_C4R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
IppStatus ippiImageRamp_32f_AC4R(Ipp32f* pDst, int DstStep, IppiSize
    roiSize, float offset, float slope, IppiAxis axis);
```

## SampleLine

Stores raster line into buffer.

```
IppStatus ippiSampleLine_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u* pDst, IppiPoint pt1, IppiPoint pt2);
IppStatus ippiSampleLine_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u* pDst, IppiPoint pt1, IppiPoint pt2);
IppStatus ippiSampleLine_16u_C1R(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, Ipp16u* pDst, IppiPoint pt1, IppiPoint pt2);
IppStatus ippiSampleLine_16u_C3R(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, Ipp16u* pDst, IppiPoint pt1, IppiPoint pt2);
```

```
IppStatus ippiSampleLine_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp32f* pDst, IppiPoint pt1, IppiPoint pt2);  
IppStatus ippiSampleLine_32f_C3R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp32f* pDst, IppiPoint pt1, IppiPoint pt2);
```

### ZigzagFwd8x8

Converts a conventional order to the zigzag order.

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

### ZigzagInv8x8

Converts a zigzag order to the conventional order.

```
IppStatus ippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

## Image Arithmetic and Logical Operations

### Arithmetic Operations

#### Add

Adds pixel values of two image buffers.

##### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiAdd_8u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiAdd_8u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiAdd_8u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiAdd_8u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiAdd_16u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiAdd_16u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const  
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize  
    roiSize, int scaleFactor);
```

```
IppStatus ippiAdd_16u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C1RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C3RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16s_AC4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_C1RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_C3RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_AC4RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_32sc_C1RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_32sc_C3RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiAdd_32sc_AC4RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
```

### **Case 2: Not-in-place operation on floating-point or complex data**

```
IppStatus ippiAdd_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
```

```
IppStatus ippiAdd_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32fc_C1R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32fc_C3R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32fc_AC4R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
```

### **Case 3: In-place operation on integer or complex data**

```
IppStatus ippiAdd_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16u_AC4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16u_C4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_C1RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_C3RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_16sc_AC4RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAdd_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAdd_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 4: In-place operation on floating-point or complex data**

```
IppStatus ippiAdd_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32f_C3IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32f_AC4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32f_C4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32fc_C1IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32fc_C3IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_32fc_AC4IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

**Case 5: In-place operation using a floating-point accumulator image**

```
IppStatus ippiAdd_8u32f_C1IR(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_8s32f_C1IR(const Ipp8s* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAdd_16u32f_C1IR(const Ipp16u* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

**Case 6: Masked in-place operation using a floating-point accumulator image**

```
IppStatus ippiAdd_8u32f_C1IMR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAdd_8s32f_C1IMR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAdd_16u32f_C1IMR(const Ipp16u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAdd_32f_C1IMR(const Ipp32f* pSrc, int srcStep, const Ipp8u*
    pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```

## AddC

Adds a constant to pixel values of an image buffer.

**Case 1: Not-in-place operation on 1-channel integer or complex data**

```
IppStatus ippiAddC_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s
    value, Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16sc_C1RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc
    value, Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc
    value, Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

**Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatus ippiAddC_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16sc_C3RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_16sc_AC4RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_32sc_C3RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiAddC_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

```
IppStatus ippiAddC_16u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u  
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int  
    scaleFactor);
```

```
IppStatus ippiAddC_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, const  
    Ipp16s value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize, int  
    scaleFactor);
```

### **Case 3: Not-in-place operation on 1-channel floating-point or complex data**

```
IppStatus ippiAddC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,  
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc  
    value, Ipp32fc* pDst, int dstStep, IppiSize roiSize);
```

### **Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiAddC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f  
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f  
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_32fc_C3R(const Ipp32fc* pSrc, int srcStep, const  
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_32fc_AC4R(const Ipp32fc* pSrc, int srcStep, const  
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f  
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

### **Case 5: In-place operation on 1-channel integer or complex data**

```
IppStatus ippiAddC_8u_C1IRSfs(Ipp8u value, Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16u_C1IRSfs(Ipp8u value, Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16s_C1IRSfs(Ipp16s value, Ipp16s* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16sc_C1IRSfs(Ipp16sc value, Ipp16sc* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_32sc_C1IRSfs(Ipp32sc value, Ipp32sc* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

### **Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatus ippiAddC_8u_C3IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_8u_AC4IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16u_C3IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16u_AC4IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int  
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiAddC_16s_C3IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,  
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

```

IppStatus ippiAddC_16s_AC4IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16sc_C3IRSfs(const Ipp16sc value[3], Ipp16sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16sc_AC4IRSfs(const Ipp16sc value[3], Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_32sc_C3IRSfs(const Ipp32sc value[3], Ipp32sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_32sc_AC4IRSfs(const Ipp32sc value[3], Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_8u_C4IRSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16u_C4IRSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiAddC_16s_C4IRSfs(const Ipp16s value[4], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);

```

#### Case 7: In-place operation on 1-channel floating-point or complex data

```

IppStatus ippiAddC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAddC_32fc_C1IR(Ipp32fc value, Ipp32fc* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

#### Case 8: In-place operation on multi-channel floating-point or complex data

```

IppStatus ippiAddC_32f_C3IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAddC_32f_AC4IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAddC_32fc_C3IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAddC_32fc_AC4IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

## AddSquare

Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.

#### Case 1: In-place operation

```

IppStatus ippiAddSquare_8u32f_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddSquare_8s32f_C1IR(const Ipp8s* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddSquare_16u32f_C1IR(const Ipp16u* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddSquare_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);

```



### Case 2: Masked in-place operation

```
IppStatus ippiAddSquare_8u32f_C1IMR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAddSquare_8s32f_C1IMR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAddSquare_16u32f_C1IMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiAddSquare_32f_C1IMR(const Ipp32f* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```

## AddProduct

Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.

### Case 1: In-place operation

```
IppStatus ippiAddProduct_8u32f_C1IR(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiAddProduct_16u32f_C1IR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiAddProduct_8s32f_C1IR(const Ipp8s* pSrc1, int src1Step,
    const Ipp8s* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiAddProduct_32f_C1IR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

### Case 2: Masked in-place operation

```
IppStatus ippiAddProduct_8u32f_C1IMR(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddProduct_8s32f_C1IMR(const Ipp8s* pSrc1, int src1Step,
    const Ipp8s* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddProduct_16u32f_C1IMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAddProduct_32f_C1IMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

## AddWeighted

Adds pixel values of a source image multiplied by factor  $a$  to floating-point pixel values of an accumulator image multiplied by factor  $(1 - a)$ .

### Case 1: In-place operation

```
IppStatus ippiAddWeighted_8u32f_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_8s32f_C1IR(const Ipp8s* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_16u32f_C1IR(const Ipp16u* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_32f_C1IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

### Case 2: Masked in-place operation

```
IppStatus ippiAddWeighted_8u32f_C1IMR(const Ipp8u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_8s32f_C1IMR(const Ipp8s* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_16u32f_C1IMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f alpha);
IppStatus ippiAddWeighted_32f_C1IMR(const Ipp32f* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f alpha);
```

## Mul

Multiplies pixel values of two image buffers.

### Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiMul_8u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_8u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_8u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_8u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
```

```
IppStatus ippiMul_16u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16s_C1RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16s_C3RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16s_AC4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16s_C4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16sc_C1RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16sc_C3RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_16sc_AC4RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_32sc_C1RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_32sc_C3RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiMul_32sc_AC4RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
```

## Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiMul_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMul_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
```

```
IppStatus ippiMul_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMul_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMul_32fc_C1R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMul_32fc_C3R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMul_32fc_AC4R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
```

### **Case 3: In-place operation on integer or complex data**

```
IppStatus ippiMul_8u_C1IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_8u_C3IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_8u_AC4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_8u_C4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16u_C1IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16u_C3IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16u_AC4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16u_C4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16s_C1IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16s_C3IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16s_AC4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16s_C4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16sc_C1IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16sc_C3IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_16sc_AC4IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiMul_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_32sc_C3RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMul_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 4: In-place operation on floating-point or complex data**

```
IppStatus ippiMul_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32f_C3IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32f_AC4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32f_C4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32fc_C1IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32fc_C3IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMul_32fc_AC4IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## MulC

Multiplies pixel values of an image buffer by a constant.

**Case 1: Not-in-place operation on 1-channel integer or complex data**

```
IppStatus ippiMulC_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s
    value, Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16sc_C1RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc
    value, Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc
    value, Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

**Case 2: Not-in-place operation on multi-channel integer or complex data**

```
IppStatus ippiMulC_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

```

IppStatus ippiMulC_16u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16sc_C3RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16sc_AC4RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_32sc_C3RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiMulC_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);

```

### Case 3: Not-in-place operation on 1-channel floating-point or complex data

```

IppStatus ippiMulC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc
    value, Ipp32fc* pDst, int dstStep, IppiSize roiSize);

```

### Case 4: Not-in-place operation on multi-channel floating-point or complex data

```

IppStatus ippiMulC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulC_32fc_C3R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);

```

```
IppStatus ippiMulC_32fc_AC4R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data**

```
IppStatus ippiMulC_8u_C1RSfs(Ipp8u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16u_C1RSfs(Ipp8u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16s_C1RSfs(Ipp16s value, Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16sc_C1RSfs(Ipp16sc value, Ipp16sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_32sc_C1RSfs(Ipp32sc value, Ipp32sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatus ippiMulC_8u_C3RSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_8u_AC4RSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16u_C3RSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16u_AC4RSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16s_C3RSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16s_AC4RSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16sc_C3RSfs(const Ipp16sc value[3], Ipp16sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16sc_AC4RSfs(const Ipp16sc value[3], Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_32sc_C3RSfs(const Ipp32sc value[3], Ipp32sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_32sc_AC4RSfs(const Ipp32sc value[3], Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_8u_C4RSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16u_C4RSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMulC_16s_C4RSfs(const Ipp16s value[4], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 7: In-place operation on 1-channel floating-point or complex data**

```
IppStatus ippiMulC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

```
IppStatus ippiMulC_32fc_C1IR(Ipp32fc value, Ipp32fc* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

### Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiMulC_32f_C3IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiMulC_32f_AC4IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiMulC_32fc_C3IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiMulC_32fc_AC4IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

## MulScale

Multiplies pixel values of two image buffers and scales the products.

### Case 1: Not-in-place operation

```
Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulScale_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_16u_C1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_16u_C3R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_16u_C4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulScale_16u_AC4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiMulScale_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```



```
IppStatus ippiMulScale_8u_C4IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_8u_AC4IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_16u_C1IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_16u_C3IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_16u_C4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulScale_16u_AC4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## MulCScale

Multiplies pixel values of an image buffer by a constant and scales the products.

### Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiMulCScale_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u
    value, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiMulCScale_8u_C3R(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_8u_AC4R(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_C3R(const Ipp16u* pSrc, int srcStep, const
    Ipp16u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_AC4R(const Ipp16u* pSrc, int srcStep, const
    Ipp16u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_8u_C4R(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_C4R(const Ipp16u* pSrc, int srcStep, const
    Ipp16u value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 3: In-place operation on 1-channel data

```
IppStatus ippiMulCScale_8u_C1IR(Ipp8u value, const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_C1IR(Ipp16u value, const Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiMulCScale_8u_C3IR(const Ipp8u value[3], const Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulCScale_8u_AC4IR(const Ipp8u value[3], const Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

```

IppStatus ippiMulCScale_16u_C3IR(const Ipp16u value[3], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_AC4IR(const Ipp16u value[3], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulCScale_8u_C4IR(const Ipp8u value[4], const Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulCScale_16u_C4IR(const Ipp16u value[4], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);

```

## Sub

Subtracts pixel values of two buffers.

### Case 1: Not-in-place operation on integer or complex data

```

IppStatus ippiSub_8u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_8u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_8u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_8u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16s_C1RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16s_C3RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16s_AC4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);

```

```
IppStatus ippiSub_16s_C4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16sc_C1RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16sc_C3RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_16sc_AC4RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_32sc_C1RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_32sc_C3RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSub_32sc_AC4RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
```

**Case 2: Not-in-place operation on floating-point or complex data**

```
IppStatus ippiSub_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32fc_C1R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32fc_C3R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiSub_32fc_AC4R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
```

**Case 3: In-place operation on integer or complex data**

```
IppStatus ippiSub_8u_C1IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

---

```

IppStatus ippiSub_8u_C3IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_8u_AC4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_8u_C4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16u_C1IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16u_C3IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16u_AC4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16u_C4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16s_C1IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16s_C3IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16s_AC4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16s_C4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16sc_C1IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16sc_C3IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_16sc_AC4IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_32sc_C1IRSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_32sc_C3IRSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSub_32sc_AC4IRSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

```

#### **Case 4: In-place operation on floating-point or complex data**

```

IppStatus ippiSub_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiSub_32f_C3IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiSub_32f_AC4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiSub_32f_C4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiSub_32fc_C1IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);

```

```
IppStatus ippiSub_32fc_C3IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiSub_32fc_AC4IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## SubC

Subtracts a constant from pixel values of an image buffer.

### Case 1: Not-in-place operation on 1-channel integer or complex data

```
IppStatus ippiSubC_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s
    value, Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16sc_C1RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc
    value, Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc
    value, Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

### Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiSubC_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16sc_C3RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16sc_AC4RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

```
IppStatus ippiSubC_32sc_C3RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiSubC_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

**Case 3: Not-in-place operation on 1-channel floating-point or complex data**

```
IppStatus ippiSubC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc
    value, Ipp32fc* pDst, int dstStep, IppiSize roiSize);
```

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiSubC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiSubC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_C3R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_AC4R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data**

```
IppStatus ippiSubC_8u_C1IRSfs(Ipp8u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16u_C1IRSfs(Ipp8u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16s_C1IRSfs(Ipp16s value, Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16sc_C1IRSfs(Ipp16sc value, Ipp16sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_32sc_C1IRSfs(Ipp32sc value, Ipp32sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 6: In-place operation on multi-channel integer or complex data**

```
IppStatus ippiSubC_8u_C3IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiSubC_8u_AC4IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16u_C3IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16u_AC4IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16s_C3IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16s_AC4IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16sc_C3IRSfs(const Ipp16sc value[3], Ipp16sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16sc_AC4IRSfs(const Ipp16sc value[3], Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_32sc_C3IRSfs(const Ipp32sc value[3], Ipp32sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_32sc_AC4IRSfs(const Ipp32sc value[3], Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_8u_C4IRSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16u_C4IRSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSubC_16s_C4IRSfs(const Ipp16s value[4], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 7: In-place operation on 1-channel floating-point or complex data**

```
IppStatus ippiSubC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_C1IR(Ipp32fc value, Ipp32fc* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiSubC_32f_C3IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiSubC_32f_AC4IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_C3IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiSubC_32fc_AC4IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiSubC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

## Div

Divides pixel values of an image buffer by pixel values of another buffer.

### Case 1: Not-in-place operation on integer or complex data

```

IppStatus ippiDiv_8u_C1RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_8u_C3RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_8u_AC4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_8u_C4RSfs(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C1RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C3RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16s_AC4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_C1RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_C3RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_AC4RSfs(const Ipp16sc* pSrc1, int src1Step, const
    Ipp16sc* pSrc2, int src2Step, Ipp16sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_32sc_C1RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_32sc_C3RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiDiv_32sc_AC4RSfs(const Ipp32sc* pSrc1, int src1Step, const
    Ipp32sc* pSrc2, int src2Step, Ipp32sc* pDst, int dstStep, IppiSize
    roiSize, int scaleFactor);

```



### Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiDiv_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32fc_C1R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32fc_C3R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiDiv_32fc_AC4R(const Ipp32fc* pSrc1, int src1Step, const
    Ipp32fc* pSrc2, int src2Step, Ipp32fc* pDst, int dstStep, IppiSize
    roiSize);
```

### Case 3: In-place operation on integer or complex data

```
IppStatus ippiDiv_8u_C1IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_8u_C3IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_8u_AC4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_8u_C4IRSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C1IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C3IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16s_AC4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16s_C4IRSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_C1IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_C3IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDiv_16sc_AC4IRSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

```

IppStatus ippDiv_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippDiv_32sc_C3RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippDiv_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

```

#### Case 4: In-place operation on floating-point or complex data

```

IppStatus ippDiv_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32f_C3IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32f_AC4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32f_C4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32fc_C1IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32fc_C3IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippDiv_32fc_AC4IR(const Ipp32fc* pSrc, int srcStep, Ipp32fc*
    pSrcDst, int srcDstStep, IppiSize roiSize);

```

## DivC

Divides pixel values of an image buffer by a constant.

#### Case 1: Not-in-place operation on 1-channel integer or complex data

```

IppStatus ippDivC_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippDivC_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s
    value, Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippDivC_16sc_C1RSfs(const Ipp16sc* pSrc, int srcStep, Ipp16sc
    value, Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippDivC_32sc_C1RSfs(const Ipp32sc* pSrc, int srcStep, Ipp32sc
    value, Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);

```

#### Case 2: Not-in-place operation on multi-channel integer or complex data

```

IppStatus ippDivC_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippDivC_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippDivC_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);

```

```
IppStatus ippiDivC_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiDivC_16sc_C3RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiDivC_16sc_AC4RSfs(const Ipp16sc* pSrc, int srcStep, const
    Ipp16sc value[3], Ipp16sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiDivC_32sc_C3RSfs(const Ipp32sc32sc* pSrc, int srcStep,
    const Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize,
    int scaleFactor);
IppStatus ippiDivC_32sc_AC4RSfs(const Ipp32sc* pSrc, int srcStep, const
    Ipp32sc value[3], Ipp32sc* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiDivC_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
IppStatus ippiDivC_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize, int
    scaleFactor);
```

**Case 3: Not-in-place operation on 1-channel floating-point or complex data**

```
IppStatus ippiDivC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc
    value, Ipp32fc* pDst, int dstStep, IppiSize roiSize);
```

**Case 4: Not-in-place operation on multi-channel floating-point or complex data**

```
IppStatus ippiDivC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDivC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_C3R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_AC4R(const Ipp32fc* pSrc, int srcStep, const
    Ipp32fc value[3], Ipp32fc* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data**

```
IppStatus ippiDivC_8u_C1IRSfs(Ipp8u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16s_C1IRSfs(Ipp16s value, Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16sc_C1IRSfs(Ipp16sc value, Ipp16sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_32sc_C1IRSfs(Ipp32sc value, Ipp32sc* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
```

**Case 6: In-place operation on multi-channel integer or complex data**

```

IppStatus ippiDivC_8u_C3IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_8u_AC4IRSfs(const Ipp8u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16s_C3IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16s_AC4IRSfs(const Ipp16s value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16sc_C3IRSfs(const Ipp16sc value[3], Ipp16sc* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16sc_AC4IRSfs(const Ipp16sc value[3], Ipp16sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_32sc_C3IRSfs(const Ipp32sc value[3], Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_32sc_AC4IRSfs(const Ipp32sc value[3], Ipp32sc*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_8u_C4IRSfs(const Ipp8u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiDivC_16s_C4IRSfs(const Ipp16s value[4], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);

```

**Case 7: In-place operation on 1-channel floating-point or complex data**

```

IppStatus ippiDivC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_C1IR(Ipp32fc value, Ipp32fc* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

**Case 8: In-place operation on multi-channel floating-point or complex data**

```

IppStatus ippiDivC_32f_C3IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiDivC_32f_AC4IR(const Ipp32f value[3], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_C3IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiDivC_32fc_AC4IR(const Ipp32fc value[3], Ipp32fc* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiDivC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

**Abs**

Computes absolute pixel values of a source image and places them into the destination image.

**Case 1: Not-in-place operation**

```

IppStatus ippiAbs_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);

```

```
IppStatus ippiAbs_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiAbs_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiAbs_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiAbs_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```

## AbsDiff

Calculates absolute difference between two images.

```
IppStatus ippiAbsDiff_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAbsDiff_16u_C1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAbsDiff_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
    IppiSize roiSize);
```

## AbsDiffC

Calculates absolute difference between image and scalar value.

```
IppStatus ippiAbsDiffC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int value);
IppStatus ippiAbsDiffC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int value);
IppStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);
```

## Sqr

Squares pixel values of an image and writes them into the destination image.

### Case 1: Not-in-place operation on integer data

```
IppStatus ippiSqr_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_8u_C4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16u_C1RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16u_C3RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16u_C4RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16u_AC4RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16s_C4RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqr_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiSqr_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiSqr_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiSqr_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

```
IppStatus ippiSqr_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,  
    int dstStep, IppiSize roiSize);
```

**Case 3: In-place operation on integer data**

```
IppStatus ippiSqr_8u_C1RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_8u_C3RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_8u_C4RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_8u_AC4RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16u_C1RSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16u_C3RSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16u_C4RSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16u_AC4RSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16s_C1RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16s_C3RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16s_C4RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiSqr_16s_AC4RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiSqr_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);  
IppStatus ippiSqr_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);  
IppStatus ippiSqr_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);  
IppStatus ippiSqr_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);
```

## Sqrt

Computes square roots of pixel values of a source image and writes them into the destination image.

**Case 1: Not-in-place operation on integer data**

```
IppStatus ippiSqrt_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize, int scaleFactor);  
IppStatus ippiSqrt_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize, int scaleFactor);
```

```
IppStatus ippiSqrt_8u_AC4RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_C1RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_C3RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_AC4RSfs(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_AC4RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

**Case 2: Not-in-place operation on floating-point data**

```
IppStatus ippiSqrt_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiSqrt_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiSqrt_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 3: In-place operation on integer data**

```
IppStatus ippiSqrt_8u_C1IRSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_8u_C3IRSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_8u_AC4IRSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_C1IRSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_C3IRSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16u_AC4IRSfs(Ipp16u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_C1IRSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_C3IRSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiSqrt_16s_AC4IRSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiSqrt_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiSqrt_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```



```
IppStatus ippiSqrt_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);  
IppStatus ippiSqrt_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);
```

## Ln

Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.

### Case 1: Not-in-place operation on integer data

```
IppStatus ippiLn_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize, int scaleFactor);  
IppStatus ippiLn_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize, int scaleFactor);  
IppStatus ippiLn_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*  
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);  
IppStatus ippiLn_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*  
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiLn_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,  
    int dstStep, IppiSize roiSize);  
IppStatus ippiLn_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,  
    int dstStep, IppiSize roiSize);
```

### Case 3: In-place operation on integer data

```
IppStatus ippiLn_8u_C1IRSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiLn_8u_C3IRSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiLn_16s_C1IRSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);  
IppStatus ippiLn_16s_C3IRSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize  
    roiSize, int scaleFactor);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiLn_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);  
IppStatus ippiLn_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize  
    roiSize);
```

## Exp

Computes the exponential of pixel values in a source image and writes the results into the destination image.

### Case 1: Not-in-place operation on integer data

```
IppStatus ippiExp_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize, int scaleFactor);
```

```

IppStatus ippiExp_8u_C3RSfs(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiExp_16s_C1RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiExp_16s_C3RSfs(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, int scaleFactor);

```

### Case 2: Not-in-place operation on floating-point data

```

IppStatus ippiExp_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
IppStatus ippiExp_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);

```

### Case 3: In-place operation on integer data

```

IppStatus ippiExp_8u_C1RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiExp_8u_C3RSfs(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiExp_16s_C1RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);
IppStatus ippiExp_16s_C3RSfs(Ipp16s* pSrcDst, int srcDstStep, IppiSize
    roiSize, int scaleFactor);

```

### Case 4: In-place operation on floating-point data

```

IppStatus ippiExp_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiExp_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize);

```

## Complement

Converts negative number from the complement to direct code.

```

IppStatus ippiComplement_32s_C1IR(Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);

```

## Logical Operations

### And

Performs a bitwise AND operation between corresponding pixels of two source buffers.

#### Case 1: Not-in-place operation

```

IppStatus ippiAnd_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAnd_8u_C3R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAnd_8u_C4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);

```

```
IppStatus ippiAnd_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAnd_16u_C1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_16u_C3R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_16u_C4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_16u_AC4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_32s_C1R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_32s_C3R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_32s_C4R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiAnd_32s_AC4R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
```

**Case 2: In-place operation**

```
IppStatus ippiAnd_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_8u_C4IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_8u_AC4IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_16u_C1IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_16u_C3IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_16u_C4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_16u_AC4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_32s_C1IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_32s_C3IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

```

IppStatus ippiAnd_32s_C4IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAnd_32s_AC4IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);

```

## AndC

Performs a bitwise AND operation of each pixel with a constant.

### Case 1: Not-in-place operation on 1-channel data

```

IppStatus ippiAndC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u value,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s value,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

### Case 2: Not-in-place operation on multi-channel data

```

IppStatus ippiAndC_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_C3R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_AC4R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_C3R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_AC4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_C4R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_C4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[4], Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

### Case 3: In-place operation on 1-channel data

```

IppStatus ippiAndC_8u_C1IR(Ipp8u value, const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_C1IR(Ipp16u value, const Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_C1IR(Ipp32s value, const Ipp32s* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

### Case 4: In-place operation on multi-channel data

```

IppStatus ippiAndC_8u_C3IR(const Ipp8u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);

```

```
IppStatus ippiAndC_8u_AC4IR(const Ipp8u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_C3IR(const Ipp16u value[3], const Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_AC4IR(const Ipp16u value[3], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_C3IR(const Ipp32s value[3], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_AC4IR(const Ipp32s value[3], const Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_8u_AC4IR(const Ipp8u value[4], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_16u_AC4IR(const Ipp16u value[4], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiAndC_32s_AC4IR(const Ipp32s value[4], const Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## Not

Performs a bitwise NOT operation on each pixel of a source buffer.

### Case 1: Not-in-place operation

```
IppStatus ippiNot_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiNot_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiNot_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize);
IppStatus ippiNot_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiNot_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiNot_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiNot_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiNot_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```

## Or

Performs bitwise inclusive OR operation between pixels of two source buffers.

### Case 1: Not-in-place operation

```
IppStatus ippiOr_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiOr_8u_C3R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_8u_C4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u*
    pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C3R(const Ipp16u* pSrc1, int src1Step, const Ipp16u*
    pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C4R(const Ipp16u* pSrc1, int src1Step, const Ipp16u*
    pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_16u_AC4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiOr_32s_C1R(const Ipp32s* pSrc1, int src1Step, const Ipp32s*
    pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_32s_C3R(const Ipp32s* pSrc1, int src1Step, const Ipp32s*
    pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_32s_C4R(const Ipp32s* pSrc1, int src1Step, const Ipp32s*
    pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOr_32s_AC4R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
```

**Case 2: In-place operation**

```
IppStatus ippiOr_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_8u_C4IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_8u_AC4IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C1IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C3IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_16u_C4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_16u_AC4IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_32s_C1IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_32s_C3IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

```
IppStatus ippiOr_32s_C4IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiOr_32s_AC4IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## OrC

Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.

### Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiOrC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u value,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s value,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiOrC_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C3R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_AC4R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C3R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_AC4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C4R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[4], Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

### Case 3: In-place operation on 1-channel data

```
IppStatus ippiOrC_8u_C1IR(Ipp8u value, const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C1IR(Ipp16u value, const Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C1IR(Ipp32s value, const Ipp32s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiOrC_8u_C3IR(const Ipp8u value[3], const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

```

IppStatus ippiOrC_8u_AC4IR(const Ipp8u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C3IR(const Ipp16u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_AC4IR(const Ipp16u value[3], const Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C3IR(const Ipp32s value[3], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_AC4IR(const Ipp32s value[3], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_8u_C4IR(const Ipp8u value[4], const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_16u_C4IR(const Ipp16u value[4], const Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiOrC_32s_C4IR(const Ipp32s value[4], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);

```

## Xor

Performs bitwise exclusive OR operation between pixels of two source buffers.

### Case 1: Not-in-place operation

```

IppStatus ippiXor_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXor_8u_C3R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXor_8u_C4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXor_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXor_16u_C1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiXor_16u_C3R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiXor_16u_C4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiXor_16u_AC4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiXor_32s_C1R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiXor_32s_C3R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);

```



```
ippiStatus ippiXor_32s_C4R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
ippiStatus ippiXor_32s_AC4R(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize);
```

**Case 2: In-place operation**

```
ippiStatus ippiXor_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
ippiStatus ippiXor_16u_C1IR(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
ippiStatus ippiXor_32s_C1IR(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

## XorC

Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.

**Case 1: Not-in-place operation on 1-channel data**

```
ippiStatus ippiXorC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u value,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u value,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s value,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Not-in-place operation on multi-channel data**

```
ippiStatus ippiXorC_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_16u_C3R(const Ipp16u* pSrc, int srcStep, const
    Ipp16u16u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_16u_AC4R(const Ipp16u* pSrc, int srcStep, const Ipp16u
    value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_32s_C3R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_32s_AC4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_16u_C4R(const Ipp16u16u* pSrc, int srcStep, const
    Ipp16u value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippiStatus ippiXorC_32s_C4R(const Ipp32s* pSrc, int srcStep, const Ipp32s
    value[4], Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

**Case 3: In-place operation on 1-channel data**

```
ippiStatus ippiXorC_8u_C1IR(Ipp8u value, const Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

```

IppStatus ippiXorC_16u_C1IR(Ipp16u value, const Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_32s_C1IR(Ipp32s value, const Ipp32s* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

#### Case 4: In-place operation on multi-channel data

```

IppStatus ippiXorC_8u_C3IR(const Ipp8u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_8u_AC4IR(const Ipp8u value[3], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_16u_C3IR(const Ipp16u value[3], const Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_16u_AC4IR(const Ipp16u value[3], const Ipp16u*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_32s_C3IR(const Ipp32s value[3], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_32s_AC4IR(const Ipp32s value[3], const Ipp32s*
    pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_8u_C4IR(const Ipp8u value[4], const Ipp8u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_16u_C4IR(const Ipp16u value[4], const Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiXorC_32s_C4IR(const Ipp32s value[4], const Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);

```

## LShiftC

Shifts bits in pixel values to the left.

#### Case 1: Not-in-place operation on 1-channel data

```

IppStatus ippiLShiftC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp32u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp32u
    value, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32u
    value, Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

#### Case 2: Not-in-place operation on multi-channel data

```

IppStatus ippiLShiftC_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp32u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C3R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_AC4R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C3R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

```
IppStatus ippiLShiftC_32s_AC4R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp32u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C4R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C4R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[4], Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

**Case 3: In-place operation on 1-channel data**

```
IppStatus ippiLShiftC_8u_C1IR(Ipp32u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C1IR(Ipp32u value, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C1IR(Ipp32u value, Ipp32s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiLShiftC_8u_C3IR(const Ipp32u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_8u_AC4IR(const Ipp32u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C3IR(const Ipp32u value[3], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_AC4IR(const Ipp32u value[3], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C3IR(const Ipp32u value[3], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_AC4IR(const Ipp32u value[3], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_8u_C4IR(const Ipp32u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_16u_C4IR(const Ipp32u value[4], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiLShiftC_32s_C4IR(const Ipp32u value[4], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

## RShiftC

Shifts bits in pixel values to the right.

**Case 1: Not-in-place operation on 1-channel data**

```
IppStatus ippiRShiftC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp32u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C1R(const Ipp8s* pSrc, int srcStep, Ipp32u
    value, Ipp8s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp32u
    value, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```

IppStatus ippiRShiftC_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp32u
    value, Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32u
    value, Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

### Case 2: Not-in-place operation on multi-channel data

```

IppStatus ippiRShiftC_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp32u
    value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C3R(const Ipp8s* pSrc, int srcStep, const Ipp32u
    value[3], Ipp8s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_AC4R(const Ipp8s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp8s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C3R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_AC4R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_C3R(const Ipp16s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_AC4R(const Ipp16s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C3R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_AC4R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[3], Ipp32s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp32u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C4R(const Ipp8s* pSrc, int srcStep, const Ipp32u
    value[4], Ipp8s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C4R(const Ipp16u* pSrc, int srcStep, const
    Ipp32u value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_C4R(const Ipp16s* pSrc, int srcStep, const
    Ipp32u value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C4R(const Ipp32s* pSrc, int srcStep, const
    Ipp32u value[4], Ipp32s* pDst, int dstStep, IppiSize roiSize);

```

### Case 3: In-place operation on 1-channel data

```

IppStatus ippiRShiftC_8u_C1IR(Ipp32u value, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C1IR(Ipp32u value, Ipp8s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C1IR(Ipp32u value, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_C1IR(Ipp32u value, Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C1IR(Ipp32u value, Ipp32s* pSrcDst, int
    srcDstStep, IppiSize roiSize);

```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiRShiftC_8u_C3IR(const Ipp32u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8u_AC4IR(const Ipp32u value[3], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C3IR(const Ipp32u value[3], Ipp8s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_AC4IR(const Ipp32u value[3], Ipp8s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C3IR(const Ipp32u value[3], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_AC4IR(const Ipp32u value[3], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_C3IR(const Ipp32u value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_AC4IR(const Ipp32u value[3], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C3IR(const Ipp32u value[3], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_AC4IR(const Ipp32u value[3], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8u_C4IR(const Ipp32u value[4], Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_8s_C4IR(const Ipp32u value[4], Ipp8s* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16u_C4IR(const Ipp32u value[4], Ipp16u* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_16s_C4IR(const Ipp32u value[4], Ipp16s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
IppStatus ippiRShiftC_32s_C4IR(const Ipp32u value[4], Ipp32s* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

## Alpha Composition

### AlphaComp

Combines two images using alpha (opacity) values of both images.

```
IppStatus ippiAlphaComp_8u_AC1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaComp_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppiAlphaType alphaType);
```

```
IppStatus ippiAlphaComp_16u_AC1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaComp_16u_AC4R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize
    roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaComp_8u_AP4R(const Ipp8u* const pSrc1[4], int
    src1Step, const Ipp8u* const pSrc2[4], int src2Step, Ipp8u* const
    pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaComp_16u_AP4R(const Ipp16u* const pSrc1[4], int
    src1Step, const Ipp16u* const pSrc2[4], int src2Step, Ipp16u* const
    pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

## AlphaCompC

Combines two images using constant alpha values.

```
IppStatus ippiAlphaCompC_8u_C1R(const Ipp8u* pSrc1, int src1Step, Ipp8u
    alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaCompC_8u_C3R(const Ipp8u* pSrc1, int src1Step, Ipp8u
    alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaCompC_8u_C4R(const Ipp8u* pSrc1, int src1Step, Ipp8u
    alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaCompC_8u_AC4R(const Ipp8u* pSrc1, int src1Step, Ipp8u
    alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
IppStatus ippiAlphaCompC_16u_C1R(const Ipp16u* pSrc1, int src1Step,
    Ipp16u alpha1, const Ipp16u* pSrc2, int src2Step, Ipp16u alpha2,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiAlphaType
    alphaType);
IppStatus ippiAlphaCompC_16u_C3R(const Ipp16u* pSrc1, int src1Step,
    Ipp16u alpha1, const Ipp16u* pSrc2, int src2Step, Ipp16u alpha2,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiAlphaType
    alphaType);
IppStatus ippiAlphaCompC_16u_C4R(const Ipp16u* pSrc1, int src1Step,
    Ipp16u alpha1, const Ipp16u* pSrc2, int src2Step, Ipp16u alpha2,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiAlphaType
    alphaType);
```

```
IppStatus ippiAlphaCompC_16u_AC4R(const Ipp16u* pSrc1, int src1Step,
    Ipp16u alpha1, const Ipp16u* pSrc2, int src2Step, Ipp16u alpha2,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiAlphaType
    alphaType);

IppStatus ippiAlphaCompC_8u_AP4R(const Ipp8u* const pSrc1[4], int
    src1Step, Ipp8u alpha1, const Ipp8u* const pSrc2[4], int src2Step,
    Ipp8u alpha2, Ipp8u* const pDst[4], int dstStep, IppiSize roiSize,
    IppiAlphaType alphaType);

IppStatus ippiAlphaCompC_16u_AP4R(const Ipp16u* const pSrc1[4], int
    src1Step, Ipp16u alpha1, const Ipp16u* const pSrc2[4], int src2Step,
    Ipp16u alpha2, Ipp16u* const pDst[4], int dstStep, IppiSize roiSize,
    IppiAlphaType alphaType);
```

## AlphaPremul

Pre-multiplies pixel values of an image by its alpha values.

### Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremul_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);

IppStatus ippiAlphaPremul_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiAlphaPremul_8u_AP4R(const Ipp8u* const pSrc[4], int
    srcStep, Ipp8u* const pDst[4], int dstStep, IppiSize roiSize);

IppStatus ippiAlphaPremul_16u_AP4R(const Ipp16u* const pSrc[4], int
    srcStep, Ipp16u* const pDst[4], int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiAlphaPremul_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);

IppStatus ippiAlphaPremul_16u_AC4IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);

IppStatus ippiAlphaPremul_8u_AC4IR(Ipp8u* const pSrcDst[4], int
    srcDstStep, IppiSize roiSize);

IppStatus ippiAlphaPremul_16u_AC4IR(Ipp16u* const pSrcDst[4], int
    srcDstStep, IppiSize roiSize);
```

## AlphaPremulC

Pre-multiplies pixel values of an image using constant alpha (opacity) values.

### Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremulC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u
    alpha, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiAlphaPremulC_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u
    alpha, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAlphaPremulC_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u
    alpha, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u
    alpha, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u alpha, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp16u alpha, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u alpha, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u alpha, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_AP4R(const Ipp8u* const pSrc[4], int
    srcStep, Ipp8u alpha, Ipp8u* const pDst[4], int dstStep, IppiSize
    roiSize);
IppStatus ippiAlphaPremulC_16u_AP4R(const Ipp16u* const pSrc[4], int
    srcStep, Ipp16u alpha, Ipp16u* const pDst[4], int dstStep, IppiSize
    roiSize);
```

### **Case 2: In-place operation**

```
IppStatus ippiAlphaPremulC_8u_C1IR(Ipp8u alpha, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_C3IR(Ipp8u alpha, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_C4IR(Ipp8u alpha, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_AC4IR(Ipp8u alpha, Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C1IR(Ipp16u alpha, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C3IR(Ipp16u alpha, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_C4IR(Ipp16u alpha, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_16u_AC4IR(Ipp16u alpha, Ipp16u* pSrcDst, int
    srcDstStep, IppiSize roiSize);
IppStatus ippiAlphaPremulC_8u_AP4IR(Ipp8u alpha, Ipp8u* const pSrcDst[4],
    int srcDstStep, IppiSize roiSize);
```



```
IppStatus ippiAlphaPremulC_16u_AP4IR(Ipp16u alpha, Ipp16u* const
    pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

## Image Color Conversion

### Color Model Conversion

#### RGBToYUV

Converts an RGB image to the YUV color model.

##### Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYUV_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYUV_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

##### Case 2: Operation on planar data

```
IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

##### Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep, IppiSize roiSize);
```

#### YUVToRGB

Converts a YUV image to the RGB color model.

##### Case 1: Operation on pixel-order data

```
IppStatus ippiYUVToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUVToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

##### Case 2: Operation on planar data

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

##### Case 3: Conversion from planar-order to pixel-order data

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## RGBToYUV422

Converts an RGB image to the YUV color model; uses 4:2:2 sampling.

### Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* const pSrc[3], int  
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 3: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst[3], IppiSize imgSize);
```

### Case 4: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 5: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],  
    IppiSize imgSize);
```

## YUV422ToRGB

Converts a YUV image that has 4:2:2 sampling format to the RGB color model.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data with ROI

```
IppStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Operation on planar data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst[3], IppiSize imgSize);
```

### Case 4: Conversion from planar-order to pixel-order data with ROI

```
IppStatus ippiYUV422ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiYUV422ToRGB_8u_P3AC4R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 5: Conversion from planar-order to pixel-order data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst, IppiSize imgSize);
```

## RGBToYUV420

Converts an RGB image to the YUV color model; uses 4:2:0 sampling.

### Case 1: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* const pSrc[3], int  
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 2: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst[3], IppiSize imgSize);
```

### Case 3: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Case 4: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],  
    IppiSize imgSize);
```

## YUV420ToRGB

Converts a YUV image that has 4:2:0 sampling format to the RGB image.

### Case 1: Operation on planar data with ROI

```
IppStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data without ROI

```
IppStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst[3], IppiSize imgSize);
```

### Case 3: Conversion from planar-order to pixel-order data with ROI

```
IppStatus ippiYUV420ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiYUV420ToRGB_8u_P3AC4R(const Ipp8u* const pSrc[3], int  
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 4: Conversion from planar-order to pixel-order data without ROI

```
IppStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* const pSrc[3], Ipp8u*  
    pDst, IppiSize imgSize);
```

## YUV420ToBGR

Converts a YUV image that has 4:2:0 sampling format to the BGR image.

```
IppStatus ippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### **YUV420ToRGB565** **YUV420ToRGB555** **YUV420ToRGB444**

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.

```
IppStatus ippiYUV420ToRGB565_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB555_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB444_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### **YUV420ToRGB565Dither** **YUV420ToRGB555Dither** **YUV420ToRGB444Dither**

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.

```
IppStatus ippiYUV420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### **BGR565ToYUV420** **BGR555ToYUV420**

Convert 16-bit BGR image to the 4:2:0 YUV image.

```
IppStatus ippiBGR565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### **YUV420ToBGR565** **YUV420ToBGR555** **YUV420ToBGR444**

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.

```
IppStatus ippiYUV420ToBGR565_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR555_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR444_8u16u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## YUV420ToBGR565Dither

## YUV420ToBGR555Dither

## YUV420ToBGR444Dither

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.

```
IppStatus ippiYUV420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## RGBToYCbCr

Converts an RGB image to the YCbCr color model.

### Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCbCr_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data

```
IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## YCbCrToRGB

Converts a YCbCr image to the RGB color model.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data

```
IppStatus ippiYCbCrToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## YCbCrToBGR

Converts a YCbCr image to the BGR color model.

```
IppStatus ippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## YCbCrToRGB565

## YCbCrToRGB555

## YCbCrToRGB444

Convert a YCbCr image to the 16-bit per pixel RGB image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCrToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## YCbCrToRGB565Dither

## YCbCrToRGB555Dither

## YCbCrToRGB444Dither

Convert a YCbCr image to the 16-bit per pixel RGB image with dithering.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444Dither_8u16u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**YCbCrToBGR565**  
**YCbCrToBGR555**  
**YCbCrToBGR444**

Convert a YCbCr image to the 16-bit per pixel BGR image.

**Case 1: Operation on pixel-order data**

```
IppStatus ippiYCbCrToBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion from planar-order to pixel-order data**

```
IppStatus ippiYCbCrToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**YCbCrToBGR565Dither**  
**YCbCrToBGR555Dither**  
**YCbCrToBGR444Dither**

Convert a YCbCr image to the 16-bit per pixel BGR image with dithering.

**Case 1: Operation on pixel-order data**

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_C3R(const Ipp8u* pSrc, int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR555Dither_8u16u_C3R(const Ipp8u* pSrc, int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR444Dither_8u16u_C3R(const Ipp8u* pSrc, int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion from planar-order to pixel-order data**

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCrToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## RGBToYCbCr422

Converts 24-bit per pixel RGB image to 16-bit per pixel YCbCr image

### Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* const pSrc[3], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## YCbCr422ToRGB

Converts 16-bit per pixel YCbCr image to 24-bit per pixel RGB image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## BGRToYCbCr422

Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## YCbCr422ToBGR

Converts 16-bit per pixel YCbCr image to 24-bit per pixel BGR image.

```
IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```



```
IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## **BGR565ToYCbCr422**

## **BGR555ToYCbCr422**

Converts 16-bit per pixel BGR image to 16-bit per pixel YCbCr image.

### **Case 1: Operation on pixel-order data**

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### **Case 2: Conversion from pixel-order to planar data**

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int  
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int  
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **RGBToCbYCr422**

## **RGBToCbYCr422Gamma**

Convert 24-bit per pixel RGB image to 16-bit per pixel CbYCr image.

```
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## **CbYCr422ToRGB**

Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.

```
IppStatus ippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## **BGRToCbYCr422**

Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.

```
IppStatus ippiBGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## **CbYCr422ToBGR**

Converts 16-bit per pixel CbYCr image to four channel BGR image.

```
IppStatus ippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## YCbCr422ToRGB565

## YCbCr422ToRGB555

## YCbCr422ToRGB444

Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToRGB565_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCr422ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## YCbCr422ToRGB565Dither

## YCbCr422ToRGB555Dither

## YCbCr422ToRGB444Dither

Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image with dithering.

### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### YCbCr422ToBGR565

### YCbCr422ToBGR555

### YCbCr422ToBGR444

Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image.

#### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR565_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCr422ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### YCbC422ToBGR565Dither

### YCbC422ToBGR555Dither

### YCbCr422ToBGR444Dither

Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image with dithering.

#### Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR565Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444Dither_8u16u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data

```
IppStatus ippiYCbCr422ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### RGBToYCbCr420

Converts an RGB image to the YCbCr color model; uses 4:2:0 sampling.

```
IppStatus ippiRGBToYCbCr420_8u_C3P3R, (const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## YCbCr420ToRGB

Converts a YCbCr image that has 4:2:0 sampling format to the RGB color model.

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## YCbCr420ToRGB565

## YCbCr420ToRGB555

## YCbCr420ToRGB444

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.

```
IppStatus ippiYCbCr420ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## YCbCr420ToRGB565Dither

## YCbCr420ToRGB555Dither

## YCbCr420ToRGB444Dither

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.

```
IppStatus ippiYCbCr420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## BGRToYCbCr420

Converts a BGR image to the YCbCr image with 4:2:0 sampling format.

```
IppStatus ippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## YCbCr420ToBGR

Converts a YCbCr image that has 4:2:0 sampling format to the BGR color model.

```
IppStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## **BGR565ToYCbCr420, BGR555ToYCbCr420**

Convert a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.

```
IppStatus ippiBGR565ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiBGR555ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **YCbCr420ToBGR565 YCbCr420ToBGR555 YCbCr420ToBGR444**

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.

```
IppStatus ippiYCbCr420ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## **YCbCr420ToBGR565Dither YCbCr420ToBGR555Dither YCbCr420ToBGR444Dither**

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.

```
IppStatus ippiYCbCr420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## **BGRToYCrCb420**

Converts a BGR image to the YCrCb image with 4:2:0 sampling format.

```
IppStatus ippiBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **BGR565ToYCrCb420** **BGR555ToYCrCb420**

Converts a 16-bit per pixel BGR image to the YCrCb image with 4:2:0 sampling format.

```
IppStatus ippiBGR565ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **BGRToYCbCr411**

Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.

```
IppStatus ippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **YCbCr411ToBGR**

Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.

```
IppStatus ippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

## **BGR565ToYCbCr411,** **BGR555ToYCbCr411**

Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:1:1 sampling format.

```
IppStatus ippiBGR565ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## **YCbCr411ToBGR565,** **YCbCr411ToBGR555**

Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.

```
IppStatus ippiYCbCr411ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## RGBToXYZ

Converts an RGB image to the XYZ color model.

```
IppStatus ippiRGBToXYZ_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToXYZ_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## XYZToRGB

Converts an XYZ image to the RGB color model.

```
IppStatus ippiXYZToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiXYZToRGB_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## RGBToLUV

Converts an RGB image to the LUV color model.

```
IppStatus ippiRGBToLUV_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToLUV_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLUV_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## LUVToRGB

Converts a LUV image to the RGB color model.

```
IppStatus ippiLUVToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLUVToRGB_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## BGRToLab

Converts a BGR image to the Lab color model.

```
IppStatus ippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
```

## LabToBGR

Converts a Lab image to the BGR color model.

```
IppStatus ippiLabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```



```
IppStatus ippiLabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize);
```

## RGBToYCC

Converts an RGB image to the YCC color model.

```
IppStatus ippiRGBToYCC_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToYCC_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize);
```

## YCCToRGB

Converts a YCC image to the RGB color model.

```
IppStatus ippiYCCToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCCToRGB_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize);
```

## RGBToHLS

Converts an RGB image to the HLS color model.

```
IppStatus ippiRGBToHLS_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHLS_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## HLSToRGB

Converts an HLS image to the RGB color model.

```
IppStatus ippiHLSToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToRGB_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
```

## BGRToHLS

Converts a BGR image to the HLS color model.

```
IppStatus ippiBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AP4C4R(const Ipp8u* const pSrc[4], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

## HLSToBGR

Converts an HLS image to the RGB color model.

```
IppStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* const pSrc[4], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## RGBToHSV

Converts an RGB image to the HSV color model.

```
IppStatus ippiRGBToHSV_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHSV_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHSV_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToHSV_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
```

## HSVToRGB

Converts an HSV image to the RGB color model.

```
IppStatus ippiHSVToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHSVToRGB_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiHSVToRGB_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHSVToRGB_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
```

## BGRToYCoCg

Converts a 24-bit BGR image to the YCoCg color model.

```
IppStatus ippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

## SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

```
IppStatus ippiSBGRToYCoCg_16s_C3P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_16s_C4P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_16s32s_C3P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp32s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_16s32s_C4P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp32s* pYCC[3], int yccStep, IppiSize roiSize);
```

## YCoCgToBGR

Converts a YCoCg image to the 24-bit BGR image.

```
IppStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## YCoCgToSBGR

Converts a YCoCg image to the 48-bit BGR image.

### Case 1: Conversion to 3-channel image

```
ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s*
    pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

### Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

## BGRToYCoCg\_Rev

Converts a 24-bit BGR image to the YCoCg-R color model.

```
IppStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

## SBGRToYCoCg\_Rev

Converts a 48-bit BGR image to the YCoCg-R color model.

```
IppStatus ippiSBGRToYCoCg_Rev_16s_C3P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_Rev_16s_C4P3R(const Ipp16s* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_Rev_16s32s_C3P3R(const Ipp16s* pBGR, int
    bgrStep, Ipp32s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiSBGRToYCoCg_Rev_16s32s_C4P3R(const Ipp16s* pBGR, int
    bgrStep, Ipp32s* pYCC[3], int yccStep, IppiSize roiSize);
```

## YCoCgToBGR\_Rev

Converts a YCoCg-R image to the 24-bit BGR image.

```
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## YCoCgToSBGR\_Rev

Converts a YCoCg-R image to the 48-bit BGR image.

### Case 1: Conversion to 3-channel image

```
ippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

### Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

## Color to Gray Scale Conversion

### RGBToGray

Converts an RGB image to gray scale using fixed transform coefficients.

```

IppStatus ippiRGBToGray_8u_C3C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_8u_AC4C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_16u_C3C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_16u_AC4C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_16s_C3C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_16s_AC4C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_32f_C3C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToGray_32f_AC4C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);

```

### ColorToGray

Converts an RGB image to gray scale using custom transform coefficients.

```

IppStatus ippiColorToGray_8u_C3C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_8u_AC4C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_16u_C3C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_16u_AC4C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_16s_C3C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_16s_AC4C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_32f_C3C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
IppStatus ippiColorToGray_32f_AC4C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);

```

## Lookup Table Conversion

### LUT

Maps an image by applying intensity transformation.

#### Case 1: Not-in-place operation on one-channel integer data

```
IppStatus ippiLUT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, const Ipp32s* pValues, const Ipp32s*
    pLevels, int nLevels);
IppStatus ippiLUT_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize, const Ipp32s* pValues, const Ipp32s*
    pLevels, int nLevels);
```

#### Case 2: Not-in-place operation on multi-channel integer data

```
IppStatus ippiLUT_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s*
    pLevels[3], int nLevels[3]);
IppStatus ippiLUT_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s*
    pLevels[4], int nLevels[4]);
IppStatus ippiLUT_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
    int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const
    Ipp32s* pLevels[4], int nLevels[4]);
```

#### Case 3: Not-in-place operation on one-channel floating-point data

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues, const Ipp32f*
    pLevels, int nLevels);
```

#### Case 4: Not-in-place operation on multi-channel floating-point data

```
IppStatus ippiLUT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[3], const
    Ipp32f* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[3], const
    Ipp32f* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[4], const
    Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data**

```
IppStatus ippiLUT_8u_C1IR(const Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
IppStatus ippiLUT_16su_C1IR(const Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

**Case 6: In-place operation on multi-channel integer data**

```
IppStatus ippiLUT_8u_C3IR(const Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiLUT_8u_AC4IR(const Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_16s_C3IR(const Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_16s_AC4IR(const Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_8u_C4IR(const Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
IppStatus ippiLUT_16s_C4IR(const Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

**Case 7: In-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

**Case 8: In-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
IppStatus ippiLUT_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
IppStatus ippiLUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int
    nLevels[4]);
```

**LUT\_Linear**

Maps an image by applying intensity transformation with linear interpolation.

**Case 1: Not-in-place operation on one-channel integer data**

```
IppStatus ippiLUT_Linear_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const
    Ipp32s* pLevels, int nLevels);
```



```
IppStatus ippiLUT_Linear_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const
    Ipp32s* pLevels, int nLevels);
```

#### **Case 2: Not-in-place operation on multi-channel integer data**

```
IppStatus ippiLUT_Linear_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp32s*
    pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const
    Ipp32s* pLevels[4], int nLevels[4]);
```

```
IppStatus ippiLUT_Linear_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const
    Ipp32s* pLevels[4], int nLevels[4]);
```

#### **Case 3: Not-in-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_Linear_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

#### **Case 4: Not-in-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_Linear_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[3], const
    Ipp32f* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f*
    pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

```
IppStatus ippiLUT_Linear_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f*
    pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

#### **Case 5: In-place operation on one-channel integer data**

```
IppStatus ippiLUT_Linear_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

```
IppStatus ippiLUT_Linear_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

#### **Case 6: In-place operation on multi-channel integer data**

```
IppStatus ippiLUT_Linear_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

```

IppStatus ippiLUT_Linear_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Linear_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Linear_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Linear_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
IppStatus ippiLUT_Linear_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);

```

#### Case 7: In-place operation on one-channel floating-point data

```

IppStatus ippiLUT_Linear_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);

```

#### Case 8: In-place operation on multi-channel floating-point data

```

IppStatus ippiLUT_Linear_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Linear_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Linear_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4],
    int nLevels[4]);

```

## LUT\_Cubic

Maps an image by applying intensity transformation with cubic interpolation.

#### Case 1: Operation on one-channel integer data

```

IppStatus ippiLUT_Cubic_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const
    Ipp32s* pLevels, int nLevels);
IppStatus ippiLUT_Cubic_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues, const
    Ipp32s* pLevels, int nLevels);

```

#### Case 2: Operation on multi-channel integer data

```

IppStatus ippiLUT_Cubic_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_Cubic_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);

```

```
IppStatus ippiLUT_Cubic_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_Cubic_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[3], const
    Ipp32s* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_Cubic_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const
    Ipp32s* pLevels[4], int nLevels[4]);
IppStatus ippiLUT_Cubic_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize roiSize, const Ipp32s* pValues[4], const
    Ipp32s* pLevels[4], int nLevels[4]);
```

**Case 3: Operation on one-channel floating-point data**

```
IppStatus ippiLUT_Cubic_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues, const
    Ipp32f* pLevels, int nLevels);
```

**Case 4: Operation on multi-channel floating-point data**

```
IppStatus ippiLUT_Cubic_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[3], const
    Ipp32f* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_Cubic_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[3], const
    Ipp32f* pLevels[3], int nLevels[3]);
IppStatus ippiLUT_Cubic_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[4], const
    Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data**

```
IppStatus ippiLUT_Cubic_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
IppStatus ippiLUT_Cubic_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

**Case 6: In-place operation on multi-channel integer data**

```
IppStatus ippiLUT_Cubic_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiLUT_Cubic_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiLUT_Cubic_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
IppStatus ippiLUT_Cubic_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

```
IppStatus ippiLUT_Cubic_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
```

```
IppStatus ippiLUT_Cubic_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

#### **Case 7: In-place operation on one-channel floating-point data**

```
IppStatus ippiLUT_Cubic_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

#### **Case 8: In-place operation on multi-channel floating-point data**

```
IppStatus ippiLUT_Cubic_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

```
IppStatus ippiLUT_Cubic_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

```
IppStatus ippiLUT_Cubic_32f_C4R(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int
    nLevels[4]);
```

## **LUTPalette**

Maps an image by applying intensity transformation in accordance with a palette table.

```
IppStatus ippiLUTPalette_8u24u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp24u* pDst, int dstStep, IppiSize roiSize, const Ipp24u* pTable,
    int nBitSize);
```

```
IppStatus ippiLUTPalette_8u32u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32u* pDst, int dstStep, IppiSize roiSize, const Ipp32u* pTable,
    int nBitSize);
```

```
IppStatus ippiLUTPalette_16u8u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pTable, int
    nBitSize);
```

```
IppStatus ippiLUTPalette_16u24u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp24u* pDst, int dstStep, IppiSize roiSize, const Ipp24u* pTable,
    int nBitSize);
```

```
IppStatus ippiLUTPalette_16u32u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp32u* pDst, int dstStep, IppiSize roiSize, const Ipp32u* pTable,
    int nBitSize);
```

## Reducing Bit Resolution

### ReduceBits

Reduces the bit resolution of an image.

#### Case 1: Operation on data of the same source and destination bit depths

```
IppStatus ippiReduceBits_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
    int levels);
IppStatus ippiReduceBits_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
    int levels);
IppStatus ippiReduceBits_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
    int levels);
IppStatus ippiReduceBits_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
    int levels);
IppStatus ippiReduceBits_16u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16u_C4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
```

#### Case 2: Operation on data of different source and destination bit depths

```
IppStatus ippiReduceBits_16u8u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
```

```
IppStatus ippiReduceBits_16u8u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16u8u_C4R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16u8u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16s8u_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16s8u_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16s8u_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_16s8u_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_32f8u_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_32f8u_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_32f8u_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_32f8u_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType
    dtype, int levels);
IppStatus ippiReduceBits_32f16u_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16u_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16u_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16u_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16s_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
```

```
IppStatus ippiReduceBits_32f16s_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16s_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
IppStatus ippiReduceBits_32f16s_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int noise,
    IppiDitherType dtype, int levels);
```

## Format Conversion

### YCbCr422

Converts 4:2:2 YCbCr image.

```
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCbCr422ToYCrCb422

Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.

```
IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCbCr422ToCbYCr422

Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.

```
IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCbCr422ToYCbCr420

Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.

#### Case 1: Operation on planar data

```
IppStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

**Case 2: Conversion from pixel-order to planar-order data**

```
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

**YCbCr422ToYCrCb420**

Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.

```
IppStatus ippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**YCbCr422ToYCbCr411**

Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

**Case 1: Operation on planar data**

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

**Case 2: Conversion from pixel-order to planar-order data**

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

**YCrCb422ToYCbCr422**

Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.

```
IppStatus ippiYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**YCrCb422ToYCbCr420**

Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.

```
IppStatus ippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```



### YCrCb422ToYCbCr411

Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.

```
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### CbYCr422ToYCbCr422

Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### CbYCr422ToYCbCr420

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

```
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

### CbYCr422ToYCrCb420

Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.

```
IppStatus ippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### CbYCr422ToYCbCr411

Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.

```
IppStatus ippiCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### YCbCr420

Converts 4:2:0 YCbCr image.

```
IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

```
IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
    Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3],
    IppiSize roiSize);
```

### **YCbCr420ToYCbCr422**

Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.

```
IppStatus ippiYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize)
```

### **YCbCr420ToYCbCr422\_Filter**

Convert 4:2:0 image to 4:2:2 image with additional filtering.

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, int layout);
```

### **YCbCr420ToCbYCr422**

Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.

```
IppStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize);
```

### **YCbCr420ToYCrCb420**

Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.

```
IppStatus ippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

### YCbCr420ToYCrCb420\_Filter

Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.

```
IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize, int layout);
```

### YCbCr420ToYCbCr411

Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

### YCrCb420ToYCbCr422

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.

```
IppStatus ippiYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCrCb420ToYCbCr422\_Filter

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.

```
IppStatus ippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### YCrCb420ToCbYCr422

Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.

```
IppStatus ippiYVToUY420_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCrCb420ToYCbCr420

Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.

```
IppStatus ippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

### YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

```
IppStatus ippYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

### YCbCr411

Converts 4:1:1 YCbCr image.

```
IppStatus ippYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCrCb, int dstCbCrStep,
    IppiSize roiSize);
IppStatus ippYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
    Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst[3], int dstStep[3],
    IppiSize roiSize);
```

### YCbCr411ToYCbCr422

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.

```
IppStatus ippYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize);
```

### YCbCr411ToYCrCb422

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

```
IppStatus ippYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### YCbCr411ToYCbCr420

Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.

```
IppStatus ippYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

## YCbCr411ToYCrCb420

Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.

```
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

## Color Twist

### ColorTwist

Applies a color twist matrix to an image with floating-point pixel values.

#### Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);

IppStatus ippiColorTwist_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);

IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
```

#### Case 2: Not-in-place operation on planar data

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* const pSrc[3], int
    srcStep, Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, const
    Ipp32f twist[3][4]);
```

#### Case 3: In-place operation on pixel-order data

```
IppStatus ippiColorTwist_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);

IppStatus ippiColorTwist_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

#### Case 4: In-place operation on planar data

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

## ColorTwist32f

Applies a color twist matrix to an image with integer pixel values.

### Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist32f_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_8s_C3R(const Ipp8s* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_8s_AC4R(const Ipp8s* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_16u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
IppStatus ippiColorTwist32f_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    twist[3][4]);
```

### Case 2: Not-in-place operation on planar data

```
IppStatus ippiColorTwist32f_8u_P3R(const Ipp8u* const pSrc[3], int
    srcStep, Ipp8u* const pDst[3], int dstStep, IppiSize roiSize, const
    Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_8s_P3R(const Ipp8s* const pSrc[3], int
    srcStep, Ipp8s* const pDst[3], int dstStep, IppiSize roiSize, const
    Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16u_P3R(const Ipp16u* const pSrc[3], int
    srcStep, Ipp16u* const pDst[3], int dstStep, IppiSize roiSize, const
    Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16s_P3R(const Ipp16s* const pSrc[3], int
    srcStep, Ipp16s* const pDst[3], int dstStep, IppiSize roiSize, const
    Ipp32f twist[3][4]);
```

### Case 3: In-place operation on pixel-order data

```
IppStatus ippiColorTwist32f_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

```
IppStatus ippiColorTwist32f_8s_C3IR(Ipp8s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_8s_AC4IR(Ipp8s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16u_AC4IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

**Case 4: In-place operation on planar data**

```
IppStatus ippiColorTwist32f_8u_IP3R(Ipp8u* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_8s_IP3R(Ipp8s* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16u_IP3R(Ipp16u* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
IppStatus ippiColorTwist32f_16s_IP3R(Ipp16s* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

## Gamma Correction

### GammaFwd

Performs gamma-correction of the source image with RGB data.

**Case 1: Not-in-place operation on integer pixel-order data**

```
IppStatus ippiGammaFwd_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiGammaFwd_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiGammaFwd_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiGammaFwd_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Not-in-place operation on integer planar data**

```
IppStatus ippiGammaFwd_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* const pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiGammaFwd_16u_P3R(const Ipp16u* const pSrc[3], int srcStep,
    Ipp16u* const pDst[3], int dstStep, IppiSize roiSize);
```

**Case 3: Not-in-place operation on floating-point pixel-order data**

```
IppStatus ippGammaFwd_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippGammaFwd_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 4: Not-in-place operation on floating-point planar data**

```
IppStatus ippGammaFwd_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data**

```
IppStatus ippGammaFwd_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippGammaFwd_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippGammaFwd_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippGammaFwd_16u_AC4IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

**Case 6: In-place operation on integer planar data**

```
IppStatus ippGammaFwd_8u_IP3R(Ipp8u* const pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
IppStatus ippGammaFwd_16u_IP3R(Ipp16u* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatus ippGammaFwd_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippGammaFwd_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 8: In-place operation on floating-point planar data**

```
IppStatus ippGammaFwd_32f_IP3R (Ipp32f* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

## **GammaInv**

Converts a gamma-corrected R'G'B' image back to the original RGB image.

**Case 1: Not-in-place operation on integer pixel-order data**

```
IppStatus ippGammaInv_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
```



```
IppStatus ippiGammaInv_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiGammaInv_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiGammaInv_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Not-in-place operation on integer planar data**

```
IppStatus ippiGammaInv_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* const pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiGammaInv_16u_P3R(const Ipp16u* const pSrc[3], int srcStep,
    Ipp16u* const pDst[3], int dstStep, IppiSize roiSize);
```

**Case 3: Not-in-place operation on floating-point pixel-order data**

```
IppStatus ippiGammaInv_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
IppStatus ippiGammaInv_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 4: Not-in-place operation on floating-point planar data**

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data**

```
IppStatus ippiGammaInv_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiGammaInv_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
IppStatus ippiGammaInv_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiGammaInv_16u_AC4IR(Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

**Case 6: In-place operation on integer planar data**

```
IppStatus ippiGammaInv_8u_IP3R(Ipp8u* const pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
IppStatus ippiGammaInv_16u_IP3R(Ipp16u* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

**Case 7: In-place operation on floating-point pixel-order data**

```
IppStatus ippiGammaInv_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

```
IppStatus ippiGammaInv_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

#### Case 8: In-place operation on floating-point planar data

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* const pSrcDst[3], int
    srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

## Threshold and Compare Operations

### Thresholding

#### Threshold

Performs thresholding of pixel values in an image buffer.

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u threshold, IppCmpOp
    ipiCmpOp);
```

```
IppStatus ippiThreshold_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold,
    IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold,
    IppCmpOp ipiCmpOp);
```

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], IppCmpOp ipiCmpOp);
```

```
IppStatus ippiThreshold_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], IppCmpOp ipiCmpOp);
```

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold, IppCmpOp ippCmpOp);
IppStatus ippiThreshold_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold, IppCmpOp ippCmpOp);
IppStatus ippiThreshold_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold, IppCmpOp ippCmpOp);
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], IppCmpOp ippCmpOp);
IppStatus ippiThreshold_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], IppCmpOp ippCmpOp);
IppStatus ippiThreshold_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], IppCmpOp ippCmpOp);
IppStatus ippiThreshold_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], IppCmpOp ippCmpOp);
IppStatus ippiThreshold_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], IppCmpOp ippCmpOp);
IppStatus ippiThreshold_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], IppCmpOp ippCmpOp);
```

## Threshold\_GT

Performs thresholding of pixel values in an image , using the comparison for “greater than”.

**Case 1: Not-in-place operation on one-channel data**

```
IppStatus ippiThreshold_GT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp8u threshold);
IppStatus ippiThreshold_GT_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold);
IppStatus ippiThreshold_GT_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold);
```

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiThreshold_GT_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_GT_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_GT_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3]);
IppStatus ippiThreshold_GT_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3]);
IppStatus ippiThreshold_GT_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3]);
```

```
IppStatus ippiThreshold_GT_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3]);
```

### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GT_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold);
IppStatus ippiThreshold_GT_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold);
IppStatus ippiThreshold_GT_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold);
```

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_GT_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_GT_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3]);
IppStatus ippiThreshold_GT_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3]);
IppStatus ippiThreshold_GT_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3]);
IppStatus ippiThreshold_GT_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3]);
```

## Threshold\_LT

Performs thresholding of pixel values in an image buffer, using the comparison for “less than”.

### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp8u threshold);
IppStatus ippiThreshold_LT_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold);
IppStatus ippiThreshold_LT_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold);
```

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_LT_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_LT_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3]);
IppStatus ippiThreshold_LT_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3]);
```

```
IppStatus ippiThreshold_LT_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3]);
IppStatus ippiThreshold_LT_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3]);
```

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_LT_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold);
IppStatus ippiThreshold_LT_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold);
IppStatus ippiThreshold_LT_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold);
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_LT_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_LT_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3]);
IppStatus ippiThreshold_LT_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3]);
IppStatus ippiThreshold_LT_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3]);
IppStatus ippiThreshold_LT_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3]);
IppStatus ippiThreshold_LT_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3]);
```

## Threshold\_Val

Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.

**Case 1: Not-in-place operation on one-channel data**

```
IppStatus ippiThreshold_Val_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, Ipp8u threshold, Ipp8u value,
    IppCmpOp ippcmpOp);
IppStatus ippiThreshold_Val_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold, Ipp16s
    value, IppCmpOp ippcmpOp);
IppStatus ippiThreshold_Val_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold, Ipp32f
    value, IppCmpOp ippcmpOp);
```

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiThreshold_Val_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3], const
    Ipp8u value[3], IppCmpOp ippcmpOp);
```

```
IppStatus ippiThreshold_Val_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    const Ipp8u value[3], IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3], IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3], IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3], IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3], IppCmpOp ipiCmpOp);
```

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_Val_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold, Ipp8u value, IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold, Ipp16s value, IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold, Ipp32f value, IppCmpOp ipiCmpOp);
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_Val_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3],
    IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3],
    IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3],
    IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3],
    IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], const Ipp32f value[3],
    IppCmpOp ipiCmpOp);
IppStatus ippiThreshold_Val_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], const Ipp32f value[3],
    IppCmpOp ipiCmpOp);
```

## Threshold\_GTVal

Performs thresholding of pixel values in an image. Pixels that are greater than threshold, are set to a specified value.

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiThreshold_GTVal_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u threshold, Ipp8u
    value);

ippStatus ippiThreshold_GTVal_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold, Ipp16s
    value);

ippStatus ippiThreshold_GTVal_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold, Ipp32f
    value);
```

### Case 2: Not-in-place operation on multi-channel data

```
ippStatus ippiThreshold_GTVal_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    const Ipp8u value[3]);

ippStatus ippiThreshold_GTVal_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    const Ipp8u value[3]);

ippStatus ippiThreshold_GTVal_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3]);

ippStatus ippiThreshold_GTVal_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3]);

ippStatus ippiThreshold_GTVal_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3]);

ippStatus ippiThreshold_GTVal_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3]);

ippStatus ippiThreshold_GTVal_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[4],
    const Ipp8u value[4]);

ippStatus ippiThreshold_GTVal_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[4], const Ipp16s value[4]);

ippStatus ippiThreshold_GTVal_32f_C1IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[4], const Ipp32f value[4]);
```

### Case 3: In-place operation on one-channel data

```
ippStatus ippiThreshold_GTVal_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold, Ipp8u value);

ippStatus ippiThreshold_GTVal_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold, Ipp16s value);
```

```
IppStatus ippiThreshold_GTVal_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold, Ipp32f value);
```

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_8u_C3R(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3]);
IppStatus ippiThreshold_GTVal_8u_AC4R(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3]);
IppStatus ippiThreshold_GTVal_16s_C3R(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3]);
IppStatus ippiThreshold_GTVal_16s_AC4R(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3]);
IppStatus ippiThreshold_GTVal_32f_C3R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_GTVal_32f_AC4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_GTVal_8u_C4R(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[4], const Ipp8u value[4]);
IppStatus ippiThreshold_GTVal_16s_C4R(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[4], const Ipp16s value[4]);
IppStatus ippiThreshold_GTVal_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[4], const Ipp32f value[4]);
```

### Threshold\_LTVal

Performs thresholding of pixel values in an image. Pixels that are less than threshold, are set to a specified value.

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u threshold, Ipp8u
    value);
IppStatus ippiThreshold_LTVal_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s threshold, Ipp16s
    value);
IppStatus ippiThreshold_LTVal_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f threshold, Ipp32f
    value);
```

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    const Ipp8u value[3]);
IppStatus ippiThreshold_LTVal_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[3],
    const Ipp8u value[3]);
IppStatus ippiThreshold_LTVal_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3]);
```



```
IppStatus ippiThreshold_LTVal_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[3], const Ipp16s value[3]);
IppStatus ippiThreshold_LTVal_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_LTVal_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_LTVal_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u threshold[4],
    const Ipp8u value[4]);
IppStatus ippiThreshold_LTVal_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    threshold[4], const Ipp16s value[4]);
IppStatus ippiThreshold_LTVal_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    threshold[4], const Ipp32f value[4]);
```

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiThreshold_LTVal_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp8u threshold, Ipp8u value);
IppStatus ippiThreshold_LTVal_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp16s threshold, Ipp16s value);
IppStatus ippiThreshold_LTVal_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f threshold, Ipp32f value);
```

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_LTVal_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3]);
IppStatus ippiThreshold_LTVal_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[3], const Ipp8u value[3]);
IppStatus ippiThreshold_LTVal_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3]);
IppStatus ippiThreshold_LTVal_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[3], const Ipp16s value[3]);
IppStatus ippiThreshold_LTVal_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_LTVal_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32 threshold[3], const Ipp32f value[3]);
IppStatus ippiThreshold_LTVal_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp8u threshold[4], const Ipp8u value[4]);
IppStatus ippiThreshold_LTVal_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp16s threshold[4], const Ipp16s value[4]);
IppStatus ippiThreshold_LTVal_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f threshold[4], const Ipp32f value[4]);
```

## Threshold\_LTValGTVal

Performs double thresholding of pixel values in an image buffer.

### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u thresholdLT, Ipp8u
    valueLT, Ipp8u thresholdGT, Ipp8u valueGT);

IppStatus ippiThreshold_LTValGTVal_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, Ipp16s
    thresholdLT, Ipp16s valueLT, Ipp16s thresholdGT, Ipp16s valueGT);

IppStatus ippiThreshold_LTValGTVal_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f32f
    thresholdLT, Ipp32f valueLT, Ipp32f thresholdGT, Ipp32f valueGT);
```

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u
    thresholdLT[3], const Ipp8u valueLT[3], const Ipp8u thresholdGT[3],
    const Ipp8u valueGT[3]);

IppStatus ippiThreshold_LTValGTVal_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u
    thresholdLT[3], const Ipp8u valueLT[3], const Ipp8u thresholdGT[3],
    const Ipp8u valueGT[3]);

IppStatus ippiThreshold_LTValGTVal_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    thresholdLT[3], const Ipp16s valueLT[3], const Ipp16s thresholdGT[3],
    const Ipp16s valueGT[3]);

IppStatus ippiThreshold_LTValGTVal_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s
    thresholdLT[3], const Ipp16s valueLT[3], const Ipp16s thresholdGT[3],
    const Ipp16s valueGT[3]);

IppStatus ippiThreshold_LTValGTVal_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    thresholdLT[3], const Ipp32f valueLT[3], const Ipp32f thresholdGT[3],
    const Ipp32f valueGT[3]);

IppStatus ippiThreshold_LTValGTVal_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    thresholdLT[3], const Ipp32f valueLT[3], const Ipp32f thresholdGT[3],
    const Ipp32f valueGT[3]);
```

### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp8u thresholdLT, Ipp8u valueLT, Ipp8u
    thresholdGT, Ipp8u valueGT);

IppStatus ippiThreshold_LTValGTVal_16s_C1IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp16s thresholdLT, Ipp16s valueLT,
    Ipp16s thresholdGT, Ipp16s valueGT);
```

```
IppStatus ippiThreshold_LTValGTVal_32f_C1IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32f thresholdLT, Ipp32f valueLT,
    Ipp32f thresholdGT, Ipp32f valueGT);
```

#### **Case 4: In-place operation on multi-channel data**

```
IppStatus ippiThreshold_LTValGTVal_8u_C3IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp8u thresholdLT[3], const Ipp8u
    valueLT[3], const Ipp8u thresholdGT[3], const Ipp8u valueGT[3]);
IppStatus ippiThreshold_LTValGTVal_8u_AC4IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp8u thresholdLT[3], const Ipp8u
    valueLT[3], const Ipp8u thresholdGT[3], const Ipp8u valueGT[3]);
IppStatus ippiThreshold_LTValGTVal_16s_C3IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp16s thresholdLT[3], const
    Ipp16s valueLT[3], const Ipp16s thresholdGT[3], const Ipp16s
    valueGT[3]);
IppStatus ippiThreshold_LTValGTVal_16s_AC4IR(Ipp16s* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp16s thresholdLT[3], const
    Ipp16s valueLT[3], const Ipp16s thresholdGT[3], const Ipp16s
    valueGT[3]);
IppStatus ippiThreshold_LTValGTVal_32f_C3IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp32f thresholdLT[3], const
    Ipp32f valueLT[3], const Ipp32f thresholdGT[3], const Ipp32f
    valueGT[3]);
IppStatus ippiThreshold_LTValGTVal_32f_AC4IR(Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize, const Ipp32f thresholdLT[3], const
    Ipp32f valueLT[3], const Ipp32f thresholdGT[3], const Ipp32f
    valueGT[3]);
```

## **Compare Operations**

### **Compare**

Compares pixel values of two images using a specified compare operation.

```
IppStatus ippiCompare_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ippCmpOp);
IppStatus ippiCompare_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ippCmpOp);
IppStatus ippiCompare_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ippCmpOp);
IppStatus ippiCompare_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ippCmpOp);
IppStatus ippiCompare_16s_C1R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ippCmpOp);
```

```

IppStatus ippiCompare_16s_C3R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_16s_C4R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_16s_AC4R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);
IppStatus ippiCompare_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, IppCmpOp ipiCmpOp);

```

## CompareC

Compares pixel values of a source image to a given value using a specified compare operation.

### Case 1: Operation on one-channel data

```

IppStatus ippiCompareC_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f
    value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);

```

### Case 2: Operation on multi-channel data

```

IppStatus ippiCompareC_8u_C3R(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_8u_AC4R(const Ipp8u* pSrc, int srcStep, const
    Ipp8u value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_16s_C3R(const Ipp16s* pSrc, int srcStep,
    const Ipp16s value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ipiCmpOp);

```

```
IppStatus ippiCompareC_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    const Ipp16s value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ipiCmpOp);
IppStatus ippiCompareC_32f_C3R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ipiCmpOp);
IppStatus ippiCompareC_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ipiCmpOp);
IppStatus ippiCompareC_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp8u
    value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_16s_C4R(const Ipp16s* pSrc, int srcStep, const
    Ipp16s value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
IppStatus ippiCompareC_32f_C4R(const Ipp32f* pSrc, int srcStep, const
    Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp
    ipiCmpOp);
```

## CompareEqualEps

Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance eps.

```
IppStatus ippiCompareEqualEps_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, Ipp32f eps);
IppStatus ippiCompareEqualEps_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, Ipp32f eps);
IppStatus ippiCompareEqualEps_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, Ipp32f eps);
IppStatus ippiCompareEqualEps_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize
    roiSize, Ipp32f eps);
```

## CompareEqualEpsC

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance eps.

### Case 1: Operation on one-channel data

```
IppStatus ippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f value, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f
    eps);
```

**Case 2: Operation on multi-channel data**

```
IppStatus ippiCompareEqualEpsC_32f_C3R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
IppStatus ippiCompareEqualEpsC_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
```

## Morphological Operations

### Dilate3x3

Performs dilation of an image using a 3x3 mask.

**Case 1: Not-in-place operation**

```
IppStatus ippiDilate3x3_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate3x3_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: In-place operation**

```
IppStatus ippiDilate3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

```
IppStatus ippiDilate3x3_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiDilate3x3_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

## Erode3x3

Performs erosion of an image using a 3x3 mask.

### Case 1: Not-in-place operation

```
IppStatus ippiErode3x3_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiErode3x3_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiErode3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiErode3x3_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

## Dilate

Performs dilation of an image using a specified mask.

### Case 1: Not-in-place operation

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiDilate_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiDilate_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
```

### Case 2: In-place operation

```
IppStatus ippiDilate_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiDilate_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiDilate_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiDilate_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
```



## Erode

Performs erosion of an image using a specified mask.

### Case 1: Not-in-place operation

```
IppStatus ippiErode_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiErode_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiErode_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiErode_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiErode_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, const char* pMask, IppiSize
    maskSize, IppiPoint anchor);
```

### Case 2: In-place operation

```
IppStatus ippiErode_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiErode_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiErode_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiErode_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize
    roiSize, const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

## MorphologyInitAlloc

Allocates and initializes morphology state structure for erosion or dilation operation.

```
IppStatus ippiMorphologyInitAlloc_8u_C1R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);

IppStatus ippiMorphologyInitAlloc_8u_C3R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);

IppStatus ippiMorphologyInitAlloc_8u_C4R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);

IppStatus ippiMorphologyInitAlloc_32f_C1R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);

IppStatus ippiMorphologyInitAlloc_32f_C3R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);

IppStatus ippiMorphologyInitAlloc_32f_C4R(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);
```

## MorphologyFree

Frees memory allocated for the morphology state structure.

```
IppStatus ippiMorphologyFree(IppiMorphState* pState);
```

## MorphologyInit

Initializes morphology state structure for erosion or dilation operation.

```
IppStatus ippiMorphologyInit_8u_C1R(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);

IppStatus ippiMorphologyInit_8u_C3R(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);

IppStatus ippiMorphologyInit_8u_C4R(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);

IppStatus ippiMorphologyInit_32f_C1R(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);

IppStatus ippiMorphologyInit_32f_C3R(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);
```

```
IppStatus ippiMorphologyInit_32f_C4R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);
```

## MorphologyGetSize

Computes the size of the morphology state structure.

```
IppStatus ippiMorphologyInit_8u_C1R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);  
IppStatus ippiMorphologyInit_8u_C3R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);  
IppStatus ippiMorphologyInit_8u_C4R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);  
IppStatus ippiMorphologyInit_32f_C1R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);  
IppStatus ippiMorphologyInit_32f_C3R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);  
IppStatus ippiMorphologyInit_32f_C4R(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, int pSize);
```

## DilateBorderReplicate

Performs dilation of an image.

```
IppStatus ippiDilateBorderReplicate_8u_C1R(const Ipp8u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);  
IppStatus ippiDilateBorderReplicate_8u_C3R(const Ipp8u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);  
IppStatus ippiDilateBorderReplicate_8u_C4R(const Ipp8u* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);  
IppStatus ippiDilateBorderReplicate_32f_C1R(const Ipp32f* pSrc, int  
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);  
IppStatus ippiDilateBorderReplicate_32f_C3R(const Ipp32f* pSrc, int  
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);  
IppStatus ippiDilateBorderReplicate_32f_C4R(const Ipp32f* pSrc, int  
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    border, IppiMorphState* pState);
```

## ErodeBorderReplicate

Performs erosion of an image.

```
IppStatus ippiErodeBorderReplicate_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);

IppStatus ippiErodeBorderReplicate_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);

IppStatus ippiErodeBorderReplicate_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);

IppStatus ippiErodeBorderReplicate_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);

IppStatus ippiErodeBorderReplicate_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);

IppStatus ippiErodeBorderReplicate_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    border, IppiMorphState* pState);
```

## MorphAdvInitAlloc

Allocates and initializes morphology state structure for advanced morphology operations.

```
IppStatus ippiMorphAdvInitAlloc_8u_C1R(IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphAdvInitAlloc_8u_C3R(IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphAdvInitAlloc_8u_C4R(IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphAdvInitAlloc_32f_C1R(IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphAdvInitAlloc_32f_C3R(IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);
```

```
IppStatus ippiMorphAdvInitAlloc_32f_C4R(IppiMorphAdvState** ppState,  
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint  
    anchor);
```

## MorphAdvFree

Frees memory allocated for the advanced morphology state structure.

```
IppStatus ippiMorphAdvFree(IppiMorphAdvState* pState);
```

## MorphAdvInit

Initializes morphology state structure for advanced morphology operations.

```
IppStatus ippiMorphologyInit_8u_C1R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiMorphologyInit_8u_C3R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiMorphologyInit_8u_C4R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiMorphologyInit_32f_C1R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiMorphologyInit_32f_C3R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiMorphologyInit_32f_C4R( IppiMorphAdvState* pState, IppiSize  
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);
```

## MorphAdvGetSize

Computes the size of the advanced morphology state structure.

```
IppStatus ippiMorphAdvGetSize_8u_C1R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_8u_C3R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_8u_C4R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_32f_C1R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_32f_C3R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_32f_C4R(IppiSize roiSize, const Ipp8u*  
    pMask, IppiSize maskSize, int* pSize);
```

## MorphOpenBorder

Performs opening of an image.

```
IppStatus ippiMorphOpenBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphOpenBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphOpenBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphOpenBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphOpenBorder_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphOpenBorder_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

## MorphCloseBorder

Performs closing of an image.

```
IppStatus ippiMorphCloseBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphCloseBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphCloseBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphCloseBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphCloseBorder_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

```
IppStatus ippiMorphCloseBorder_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

## MorphTophatBorder

Performs top-hat operation on an image.

```
IppStatus ippiMorphTophatBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphTophatBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphTophatBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphTophatBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphTophatBorder_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphTophatBorder_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

## MorphBlackhatBorder

Performs black-hat operation on an image.

```
IppStatus ippiMorphBlackhatBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphBlackhatBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphBlackhatBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);

IppStatus ippiMorphBlackhatBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

```
IppStatus ippiMorphBlackhatBorder_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphBlackhatBorder_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

## MorphGradientBorder

Calculates morphological gradient of an image.

```
IppStatus ippiMorphGradientBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphGradientBorder_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphGradientBorder_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphGradientBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphGradientBorder_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
IppStatus ippiMorphGradientBorder_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphAdvState* pState);
```

## MorphGrayInitAlloc

Allocates and initializes morphology state structure for gray-kernel morphology operations.

```
IppStatus ippiMorphGrayInitAlloc_8u_C1R(IppiMorphGrayState_8u** ppState,
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint
    anchor);
IppStatus ippiMorphGrayInitAlloc_32f_C1R(IppiMorphGrayState_32f**
    ppState, IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize,
    IppiPoint anchor);
```



## MorphGrayFree

Frees memory allocated for the gray-kernel morphology state structure.

```
IppStatus ippiMorphGrayFree_8u_C1R(IppiMorphAdvState_8u* pState);  
IppStatus ippiMorphGrayFree_32f_C1R(IppiMorphAdvState_32f* pState);
```

## MorphGrayInit

Initializes morphology state structure for gray-kernel morphology operations.

```
IppStatus ippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState,  
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint  
    anchor);  
IppStatus ippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState,  
    IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, IppiPoint  
    anchor);
```

## MorphGrayGetSize

Computes the size of the gray-kernel morphology state structure.

```
IppStatus ippiMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s*  
    pMask, IppiSize maskSize, int* pSize);  
IppStatus ippiMorphAdvGetSize_32f_C1R(IppiSize roiSize, const Ipp32f*  
    pMask, IppiSize maskSize, int* pSize);
```

## MorphDilateBorder

Performs gray-kernel dilation of an image.

```
IppStatus ippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    borderType, IppiMorphGrayState_8u* pState);  
IppStatus ippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    borderType, IppiMorphGrayState_32f* pState);
```

## GrayErodeBorder

Performs gray-kernel erosion of an image.

```
IppStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    borderType, IppiMorphGrayState_8u* pState);  
IppStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    borderType, IppiMorphGrayState_32f* pState);
```

## MorphReconstructGetBufferSize

Computes the size of the buffer for morphological reconstruction operation.

```
IppStatus ippiMorphReconstructGetBufferSize_8u_C1(IppiSize roiSize, int*
    pSize);

IppStatus ippiMorphReconstructGetBufferSize_32f_C1(IppiSize roiSize,
    int* pSize);
```

## MorphReconstructDilate

Reconstructs an image by dilation.

```
IppStatus ippiMorphReconstructDilate_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuffer, IppiNorm norm);

IppStatus ippiMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuffer, IppiNorm norm);
```

## MorphReconstructErode

Reconstructs an image by erosion.

```
IppStatus ippiMorphReconstructErode_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuf, IppiNorm norm);

IppStatus ippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuf, IppiNorm norm);
```

# Filtering Functions

## FilterBox

Blurs an image using a simple box filter.

### Case 1: Not-in-place operation

```
IppStatus ippiFilterBox_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterBox_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterBox_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
```

```
IppStatus ippiFilterBox_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterBox_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
```

### Case 2: In-place operation

```
IppStatus ippiFilterBox_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_8u_AC4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_16s_AC4IR(Ipp16s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
IppStatus ippiFilterBox_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
```

```
IppStatus ippiFilterBox_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,  
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiFilterBox_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,  
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);  
IppStatus ippiFilterBox_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,  
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
```

## SumWindowRow

Sums pixel values in the row mask applied to the image.

```
IppStatus ippiSumWindowRow_8u32f_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);  
IppStatus ippiSumWindowRow_8u32f_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);  
IppStatus ippiSumWindowRow_8u32f_C4R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);  
IppStatus ippiSumWindowRow_16s32f_C1R(const Ipp16s* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);  
IppStatus ippiSumWindowRow_16s32f_C3R(const Ipp16s* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);  
IppStatus ippiSumWindowRow_16s32f_C4R(const Ipp16s* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    xAnchor);
```

## SumWindowColumn

Sums pixel values in the column mask applied to the image.

```
IppStatus ippiSumWindowColumn_8u32f_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    yAnchor);  
IppStatus ippiSumWindowColumn_8u32f_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    yAnchor);  
IppStatus ippiSumWindowColumn_8u32f_C4R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int  
    yAnchor);
```

```
IppStatus ippiSumWindowColumn_16s32f_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int
    yAnchor);

IppStatus ippiSumWindowColumn_16s32f_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int
    yAnchor);

IppStatus ippiSumWindowColumn_16s32f_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int
    yAnchor);
```

## FilterMin

Applies the 'min' filter to an image.

```
IppStatus ippiFilterMin_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);

IppStatus ippiFilterMin_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
```

```
IppStatus ippiFilterMin_32f_AC4R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);
```

## FilterMax

Applies the 'max' filter to an image.

```
IppStatus ippiFilterMax_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_8u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_8u_C4R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_8u_AC4R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_16s_C1R(const Ipp16s* pSrc, int srcStep,  
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_16s_C3R(const Ipp16s* pSrc, int srcStep,  
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_16s_C4R(const Ipp16s* pSrc, int srcStep,  
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_16s_AC4R(const Ipp16s* pSrc, int srcStep,  
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_32f_C3R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_32f_C4R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);  
IppStatus ippiFilterMax_32f_AC4R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,  
    IppiPoint anchor);
```

### FilterMinGetBufferSize

Computes the size of the working buffer for the minimum filter.

```
IppStatus ippiFilterMinGetBufferSize_8u_C1R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMinGetBufferSize_8u_C3R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMinGetBufferSize_8u_C4R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMinGetBufferSize_32f_C1R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMinGetBufferSize_32f_C3R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMinGetBufferSize_32f_C4R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
```

### FilterMaxGetBufferSize

Computes the size of the working buffer for the maximum filter.

```
IppStatus ippiFilterMaxGetBufferSize_8u_C1R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMaxGetBufferSize_8u_C3R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMaxGetBufferSize_8u_C4R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMaxGetBufferSize_32f_C1R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMaxGetBufferSize_32f_C3R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
IppStatus ippiFilterMaxGetBufferSize_32f_C4R(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
```

### FilterMinBorderReplicate

Applies the 'min' filter with border replication to an image.

```
IppStatus ippiFilterMinBorderReplicate_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);
IppStatus ippiFilterMinBorderReplicate_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterMinBorderReplicate_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

## FilterMaxBorderReplicate

Applies the 'max' filter with border replication to an image.

```
IppStatus ippiFilterManBorderReplicate_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);

IppStatus ippiFilterManBorderReplicate_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);

IppStatus ippiFilterManBorderReplicate_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);

IppStatus ippiFilterManBorderReplicate_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);

IppStatus ippiFilterManBorderReplicate_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);

IppStatus ippiFilterManBorderReplicate_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

## Median Filters

### FilterMedian

Filters an image using a median filter.

```
IppStatus ippiFilterMedian_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiFilterMedian_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiFilterMedian_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor);
```



```
IppStatus ippiFilterMedian_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor);
IppStatus ippiFilterMedian_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterMedian_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterMedian_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
IppStatus ippiFilterMedian_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor);
```

## FilterMedianHoriz

Filters an image using a horizontal median filter.

```
IppStatus ippiFilterMedianHoriz_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_16s_C4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianHoriz_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

## FilterMedianVert

Filters an image using a vertical median filter.

```
IppStatus ippiFilterMedianVert_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_16s_C4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianVert_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

## FilterMedianCross

Filters an image using a cross median filter.

```
IppStatus ippiFilterMedianCross_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianCross_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianCross_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianCross_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianCross_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

```
IppStatus ippiFilterMedianCross_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

## FilterMedianColor

Filters an image using a color median filter.

```
IppStatus ippiFilterMedianColor_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianColor_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianColor_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianColor_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianColor_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
IppStatus ippiFilterMedianColor_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

## General Linear Filters

### Filter

Filters an image using a general rectangular kernel.

#### Case 1: Operation on integer data

```
IppStatus ippiFilter_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize
    kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize
    kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize
    kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, IppiSize
    kernelSize, IppiPoint anchor, int divisor);
```

```

IppStatus ippiFilter_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);
IppStatus ippiFilter_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);

```

### Case 2: Operation on floating-point data

```

IppStatus ippiFilter_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);

```

## Filter32f

Filters an image with integer data using a floating-point rectangular kernel.

```

IppStatus ippiFilter32f_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);

```

```
IppStatus ippiFilter32f_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
IppStatus ippiFilter32f_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
```

## Separable Filters

### FilterColumn

Filters an image using a spatial kernel that consists of a single column.

#### Case 1: Operation on integer data

```
IppStatus ippiFilterColumn_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, int kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, int kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, int kernelSize, int yAnchor, int divisor);
IppStatus ippiFilterColumn_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, int kernelSize, int yAnchor, int divisor);
```

#### Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumn_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
IppStatus ippiFilterColumn_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
```

```
IppStatus ippiFilterColumn_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
IppStatus ippiFilterColumn_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
```

## FilterColumn32f

Filters an image with integer data using a floating-point column kernel.

```
IppStatus ippiFilterColumn32f_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
IppStatus ippiFilterColumn32f_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int yAnchor);
```

## FilterRow

Filters an image using a spatial kernel that consists of a single row.

### Case 1: Operation on integer data

```
IppStatus ippiFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
```

```
IppStatus ippiFilterRow_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
IppStatus ippiFilterRow_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s* pKernel, int
    kernelSize, int xAnchor, int divisor);
```

### Case 2: Operation on floating-point data

```
IppStatus ippiFilterRow_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
```

## FilterRow32f

Filters an image with integer data using a floating-point row kernel.

```
IppStatus ippiFilterRow32f_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int xAnchor);
```

```
IppStatus ippiFilterRow32f_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int xAnchor);
IppStatus ippiFilterRow32f_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f*
    pKernel, int kernelSize, int xAnchor);
```

### FilterRowBorderPipelineGetBufferSize

Compute the size of working buffer for the row filter.

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_8u16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_8u16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_32f_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_32f_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

### FilterRowBorderPipelineGetBufferSize\_Low

Compute the size of working buffer for the Low-flavor of row filters.

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

### FilterColumnPipelineGetBufferSize

Compute the size of working buffer for the column filter.

```
IppStatus ippiFilterColumnPipelineGetBufferSize_8u16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterColumnPipelineGetBufferSize_8u16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```



```
IppStatus ippiFilterColumnPipelineGetBufferSize_16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterColumnPipelineGetBufferSize_16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterColumnPipelineGetBufferSize_32f_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterColumnPipelineGetBufferSize_32f_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

## FilterColumnPipelineGetBufferSize\_Low

Compute the size of working buffer for the Low-flavor of column filters.

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_16s_C1R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_16s_C3R(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

## FilterRowBorderPipeline

Applies the filter with border to image rows.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp8u
    borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterRowBorderPipeline_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue, int divisor, Ipp8u* pBuffer);
```

### Case 2: Operation on one-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue, Ipp8u* pBuffer);
```

### Case 3: Operation on three-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_8u16s_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp8u
    borderValue[3], int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterRowBorderPipeline_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue[3], int divisor, Ipp8u* pBuffer);

IppStatus ippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue[3], int divisor, Ipp8u* pBuffer);
```

**Case 4: Operation on three-channel floating point data**

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue[3], Ipp8u* pBuffer);
```

## FilterRowBorderPipeline\_Low

Applies the Low-flavor filter with border to image rows.

**Case 1: Operation on one-channel data**

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
    int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue, int divisor, Ipp8u* pBuffer);
```

**Case 2: Operation on three-channel data**

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc,
    int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
    int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue[3], int divisor, Ipp8u* pBuffer);
```

## FilterColumnPipeline

Applies the filter to image columns.

**Case 1: Operation on integer data**

```
IppStatus ippiFilterColumnPipeline_16s8u_C1R(const Ipp16s** ppSrc,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp8u* pKernel,
    int kernelSize, int divisor, Ipp8u* pBuffer);

IppStatus ippiFilterColumnPipeline_16s8u_C3R(const Ipp16s** ppSrc,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp8u* pKernel,
    int kernelSize, int divisor, Ipp8u* pBuffer);

IppStatus ippiFilterColumnPipeline_16s_C1R(const Ipp16s** ppSrc, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel, int
    kernelSize, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterColumnPipeline_16s_C3R(const Ipp16s** ppSrc, Ipp16s*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel, int
    kernelSize, int divisor, Ipp8u* pBuffer);
```

### **Case 2: Operation on floating-point data**

```
IppStatus ippiFilterColumnPipeline_32f_C1R(const Ipp32f** ppSrc, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterColumnPipeline_32f_C3R(const Ipp32f** ppSrc, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int
    kernelSize, Ipp8u* pBuffer);
```

## **FilterColumnPipeline\_Low**

Applies the Low-flavor filter to image columns.

```
IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
    pKernel, int kernelSize, int divisor, Ipp8u* pBuffer, Ipp8u*
    pBuffer);
```

```
IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
    pKernel, int kernelSize, int divisor, Ipp8u* pBuffer, Ipp8u*
    pBuffer);
```

## **Wiener Filters**

### **FilterWienerGetBufferSize**

Computes the size of the external buffer for ippiFilterWiener function.

```
IppStatus ippiFilterWienerGetBufferSize(IppiSize dstRoiSize, IppiSize
    maskSize, int channels, int* pBufferSize);
```

### **FilterWiener**

Filters an image using the Wiener algorithm.

#### **Case 1: Operation on one-channel images**

```
IppStatus ippiFilterWiener_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor, Ipp32f noise, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterWiener_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterWiener_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise, Ipp8u* pBuffer);
```

### Case 2: Operation on multi-channel images

```
IppStatus ippiFilterWiener_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint
    anchor, Ipp32f noise[4], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);
IppStatus ippiFilterWiener_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
    IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);
```

## 2D Convolution

### ConvFull

Performs a full convolution of two images.

#### Case 1: Operation on integer data

```
IppStatus ippiConvFull_8u_C1R(const Ipp8u* pSrc1, int src1Step, IppiSize
    src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize src2Size, Ipp8u*
    pDst, int dstStep, int divisor);
IppStatus ippiConvFull_8u_C3R(const Ipp8u* pSrc1, int src1Step, IppiSize
    src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize src2Size, Ipp8u*
    pDst, int dstStep, int divisor);
```

```
IppStatus ippiConvFull_8u_AC4R(const Ipp8u* pSrc1, int src1Step, IppiSize
    src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize src2Size, Ipp8u*
    pDst, int dstStep, int divisor);
IppStatus ippiConvFull_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
IppStatus ippiConvFull_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
IppStatus ippiConvFull_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
```

### **Case 2: Operation on floating-point data**

```
IppStatus ippiConvFull_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
IppStatus ippiConvFull_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
IppStatus ippiConvFull_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
```

## **ConvValid**

Performs a valid convolution of two images.

### **Case 1: Operation on integer data**

```
IppStatus ippiConvValid_8u_C1R(const Ipp8u* pSrc1, int src1Step, IppiSize
    src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize src2Size, Ipp8u*
    pDst, int dstStep, int divisor);
IppStatus ippiConvValid_8u_C3R(const Ipp8u* pSrc1, int src1Step, IppiSize
    src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize src2Size, Ipp8u*
    pDst, int dstStep, int divisor);
IppStatus ippiConvValid_8u_AC4R(const Ipp8u* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp8u* pSrc2, int src2Step, IppiSize
    src2Size, Ipp8u* pDst, int dstStep, int divisor);
IppStatus ippiConvValid_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
IppStatus ippiConvValid_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
IppStatus ippiConvValid_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp16s* pSrc2, int src2Step, IppiSize
    src2Size, Ipp16s* pDst, int dstStep, int divisor);
```

**Case 2: Operation on floating-point data**

```
IppStatus ippiConvValid_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
IppStatus ippiConvValid_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
IppStatus ippiConvValid_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize
    src2Size, Ipp32f* pDst, int dstStep);
```

## Deconvolution

### DeconvFFTInitAlloc

Allocates and initializes state structure for FFT deconvolution.

```
IppStatus ippiDeconvFFTInitAlloc_32f_C1R(IppiDeconvFFTState_32f_C1R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTOrder, Ipp32f threshold);
IppStatus ippiDeconvFFTInitAlloc_32f_C3R(IppiDeconvFFTState_32f_C3R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTOrder, Ipp32f threshold);
```

### DeconvFFTFree

Frees memory allocated for the FFT deconvolution state structure.

```
IppStatus ippiDeconvFFTFree_32f_C1R(IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState)
IppStatus ippiDeconvFFTFree_32f_C3R(IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

### DeconvFFT

Performs FFT deconvolution of an image.

```
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState);
IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

## DeconvLRInitAlloc

Allocates and initializes state structure for LR deconvolution.

```
IppStatus ippiDeconvLRInitAlloc_32f_C1R(IppiDeconvLR_32f_C1R**
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,
    Ipp32f threshold);

IppStatus ippiDeconvLRInitAlloc_32f_C3R(IppiDeconvLR_32f_C3R**
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,
    Ipp32f threshold);
```

## DeconvLRFree

Frees memory allocated for the LR deconvolution state structure.

```
IppStatus ippiDeconvLRFree_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR)

IppStatus ippiDeconvLRFree_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR);
```

## DeconvLR

Performs LR deconvolution of an image.

```
IppStatus ippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C1R* pDeconvLR);

IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C3R* pDeconvLR);
```

## Fixed Filters

### FilterPrewittHoriz

Filters an image using a horizontal Prewitt kernel.

```
IppStatus ippiFilterPrewittHoriz_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
```

```
IppStatus ippiFilterPrewittHoriz_16s_C4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittHoriz_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterPrewittVert

Filters an image using a vertical Prewitt kernel.

```
IppStatus ippiFilterPrewittVert_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterPrewittVert_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterScharrHoriz

Filters an image using a horizontal Scharr kernel.

```
IppStatus ippiFilterScharrHoriz_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
```



```
IppStatus ippiFilterScharrHoriz_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterScharrHoriz_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterScharrVert

Filters an image using a vertical Scharr kernel.

```
IppStatus ippiFilterScharrVert_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterScharrVert_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterScharrVert_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterSobelHoriz, FilterSobelHorizMask

Filters an image using a horizontal Sobel kernel.

```
IppStatus ippiFilterSobelHoriz_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_8u_C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_16s_C4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelHoriz_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterSobelHoriz_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

---

```
IppStatus ippiFilterSobelHorizMask_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
```

## FilterSobelVert, FilterSobelVertMask

Filter an image using a vertical Sobel kernel.

```
IppStatus ippiFilterSobelVert_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_8u_C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_16s_C4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_32f_C4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSobelVert_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterSobelVert_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterSobelMaskVert_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
```

## FilterSobelHorizSecond

Filters an image using a second derivative horizontal Sobel operator.

```
IppStatus ippiFilterSobelHorizSecond_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
```

```
IppStatus ippiFilterSobelHorizSecond_8s16s_C1R(const Ipp8s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
IppStatus ippiFilterSobelHorizSecond_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
```

## FilterSobelVertSecond

Filters an image using a second derivative vertical Sobel operator.

```
IppStatus ippiFilterSobelVertSecond_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
IppStatus ippiFilterSobelVertSecond_8s16s_C1R(const Ipp8s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
IppStatus ippiFilterSobelVertSecond_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize
    mask);
```

## FilterSobelCross

Filters an image using a second cross derivative Sobel operator.

```
IppStatus ippiFilterSobelCross_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterSobelCross_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterSobelCross_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## FilterRobertsDown

Filters an image using a horizontal Roberts kernel.

```
IppStatus ippiFilterRobertsDown_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_8u_C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

```
IppStatus ippiFilterRobertsDown_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsDown_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterRobertsUp

Filters an image using a vertical Roberts kernel.

```
IppStatus ippiFilterRobertsUp_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_8u_AC4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_16s_C3R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_16s_AC4R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterRobertsUp_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## FilterLaplace

Filters an image using a Laplacian kernel.

```
IppStatus ippiFilterLaplace_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

```
IppStatus ippiFilterLaplace_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLaplace_8s16s_C1R(const Ipp8s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## FilterGauss

Filters an image using a Gaussian kernel.

```
IppStatus ippiFilterGauss_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterGauss_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## FilterHipass

Filters an image using a highpass filter.

```
IppStatus ippiFilterHipass_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

```
IppStatus ippiFilterHipass_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterHipass_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## FilterLowpass

Filters an image using a lowpass filter.

```
IppStatus ippiFilterLowpass_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
IppStatus ippiFilterLowpass_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

## FilterSharpen

Filters an image using a sharpening filter.

```
IppStatus ippiFilterSharpen_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_16s_C3R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_16s_C4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiFilterSharpen_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
```

## Fixed Filters with Border

### FilterScharrHorizGetBufferSize

Computes the size of the external buffer for the horizontal Scharr filter with border.

```
IppStatus ippiFilterScharrHorizGetBufferSize_8u16s_C1R(IppiSize roiSize,
    int* pBufferSize);
IppStatus ippiFilterScharrHorizGetBufferSize_32f_C1R(IppiSize roiSize,
    int* pBufferSize);
```

### FilterScharrVertGetBufferSize

Computes the size of the external buffer for the vertical Scharr filter with border.

```
IppStatus ippiFilterScharrVertGetBufferSize_8u16s_C1R(IppiSize roiSize,
    int* pBufferSize);
```

```
IppStatus ippiFilterScharrVertGetBufferSize_32f_C1R(IppiSize roiSize,  
int* pBufferSize);
```

### **FilterSobelHorizGetBufferSize**

Computes the size of the external buffer for the horizontal Sobel filter with border.

```
IppStatus ippiFilterSobelHorizGetBufferSize_8u16s_C1R(IppiSize roiSize,  
IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelHorizGetBufferSize_32f_C1R(IppiSize roiSize,  
IppiMaskSize mask, int* pBufferSize);
```

### **FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize**

Computes the size of the external buffer for the vertical Sobel filter with border.

```
IppStatus ippiFilterSobelVertGetBufferSize_8u16s_C1R(IppiSize roiSize,  
IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelVertGetBufferSize_32f_C1R(IppiSize roiSize,  
IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelNegVertGetBufferSize_8u16s_C1R(IppiSize  
roiSize, IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelNegVertGetBufferSize_32f_C1R(IppiSize roiSize,  
IppiMaskSize mask, int* pBufferSize);
```

### **FilterSobelHorizSecondGetBufferSize**

Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.

```
IppStatus ippiFilterSobelHorizSecondGetBufferSize_8u16s_C1R(IppiSize  
roiSize, IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelHorizSecondGetBufferSize_32f_C1R(IppiSize  
roiSize, IppiMaskSize mask, int* pBufferSize);
```

### **FilterSobelVertSecondGetBufferSize**

Computes the size of the external buffer for the second derivative vertical Sobel filter with border.

```
IppStatus ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R(IppiSize  
roiSize, IppiMaskSize mask, int* pBufferSize);
```



```
IppStatus ippiFilterSobelVertSecondGetBufferSize_32f_C1R(IppiSize  
    roiSize, IppiMaskSize mask, int* pBufferSize);
```

### FilterSobelCrossGetBufferSize

Computes the size of the external buffer for the cross Sobel filter with border.

```
IppStatus ippiFilterSobelCrossGetBufferSize_8u16s_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterSobelCrossGetBufferSize_32f_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);
```

### FilterLaplacianGetBufferSize

Computes the size of the external buffer for the Laplace filter with border.

```
IppStatus ippiFilterLaplacianGetBufferSize_8u16s_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterLaplacianGetBufferSize_32f_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);
```

### FilterLowpassGetBufferSize

Computes the size of the external buffer for the lowpass filter with border.

```
IppStatus ippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);  
  
IppStatus ippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize,  
    IppiMaskSize mask, int* pBufferSize);
```

### GenSobelKernel

Computes kernel for the Sobel filter.

```
IppStatus ippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx,  
    int sign);  
  
IppStatus ippiGenSobelKernel_32f(Ipp132f* pDst, int kernelSize, int dx,  
    int sign);
```

### FilterScharrHorizBorder

Applies horizontal Scharr filter with border.

```
IppStatus ippiFilterScharrHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int  
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType  
    borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterScharrHorizBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### **FilterScharrVertBorder**

Applies vertical Scharr filter with border.

```
IppStatus ippiFilterScharrVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterScharrVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### **FilterSobelHorizBorder**

Applies horizontal Sobel filter with border.

```
IppStatus ippiFilterSobelHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelHorizBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### **FilterSobelVertBorder, FilterSobelNegVertBorder**

Applies vertical Sobel filter with border.

```
IppStatus ippiFilterSobelVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelNegVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelNegVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### FilterSobelHorizSecondBorder

Applies horizontal (second derivative) Sobel filter with border.

```
IppStatus ippiFilterSobelHorizSecondBorder_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue,
    Ipp8u* pBuffer);

IppStatus ippiFilterSobelHorizSecondBorder_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue,
    Ipp8u* pBuffer);
```

### FilterSobelVertSecondBorder

Applies vertical (second derivative) Sobel filter with border.

```
IppStatus ippiFilterSobelVertSecondBorder_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue,
    Ipp8u* pBuffer);

IppStatus ippiFilterSobelVertSecondBorder_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue,
    Ipp8u* pBuffer);
```

### FilterSobelCrossBorder

Applies second derivative cross Sobel filter with border.

```
IppStatus ippiFilterSobelCrossBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelCrossBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### FilterLaplacianBorder

Applies Laplacian filter with border.

```
IppStatus ippiFilterLaplacianBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLaplacianBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

## FilterLowpassBorder

Applies lowpass filter with border.

```
IppStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

# Image Linear Transforms

## Fourier Transforms

### FFTInitAlloc

Allocates memory and fills in context data needed for the image FFT functions to operate.

```
IppStatus ippiFFTInitAlloc_R_32s (IppiFFTSpec_R_32s** pFFTSpec, int
    orderX, int orderY, int flag, IppHintAlgorithm hint);

IppStatus ippiFFTInitAlloc_R_32f (IppiFFTSpec_R_32f** pFFTSpec, int
    orderX, int orderY, int flag, IppHintAlgorithm hint);

IppStatus ippiFFTInitAlloc_C_32fc (IppiFFTSpec_C_32fc** pFFTSpec, int
    orderX, int orderY, int flag, IppHintAlgorithm hint);
```

### FFTFree

Deallocates memory used by the FFT context structure.

```
IppStatus ippiFFTFree_R_32s (IppiFFTSpec_R_32s* pFFTSpec);
IppStatus ippiFFTFree_R_32f (IppiFFTSpec_R_32f* pFFTSpec);
IppStatus ippiFFTFree_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec);
```

### FFTGetBufSize

Determines the size of external work buffer that can be used by the FFT functions.

```
IppStatus ippiFFTGetBufSize_R_32s (IppiFFTSpec_R_32s* pFFTSpec, int*
    pSize);
IppStatus ippiFFTGetBufSize_R_32f (IppiFFTSpec_R_32f* pFFTSpec, int*
    pSize);
IppStatus ippiFFTGetBufSize_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec, int*
    pSize);
```

## FFTFwd

Applies forward Fast Fourier Transform to an image.

### Case 1: Operation on an image with integer data

```
IppStatus ippiFFTFwd_RToPack_8u32s_C1RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s*
    pFFTSpec, int scaleFactor, Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_8u32s_C3RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s*
    pFFTSpec, int scaleFactor, Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_8u32s_C4RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s*
    pFFTSpec, int scaleFactor, Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_8u32s_AC4RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s*
    pFFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

### Case 2: Operation on an image with floating-point data

```
IppStatus ippiFFTFwd_RToPack_32f_C1R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_32f_C3R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_32f_C4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_32f_AC4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);
```

### Case 3: Operation on an image with complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,
    Ipp8u* pBuffer);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_32f_C1IR (Ipp32f* pSrcDst, int srcDstStep,
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippiFFTFwd_RToPack_32f_C3IR (Ipp32f* pSrcDst, int srcDstStep,
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiFFTFwd_RToPack_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);  
IppStatus ippiFFTFwd_RToPack_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

**Case 5: In-place operation on complex data**

```
IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

## FFTInv

Applies an inverse FFT to complex source data and stores results in a destination image.

**Case 1: Operation on an image with integer data**

```
IppStatus ippiFFTInv_PackToR_32s8u_C1RSfs (const Ipp32s* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32s8u_C3RSfs (const Ipp32s* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32s8u_C4RSfs (const Ipp32s* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32s8u_AC4RSfs (const Ipp32s* pSrc, int  
    srcStep, Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);
```

**Case 2: Operation on an image with floating-point data**

```
IppStatus ippiFFTInv_PackToR_32f_C1R (const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,  
    Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_C3R (const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,  
    Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_C4R (const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,  
    Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_AC4R (const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,  
    Ipp8u* pBuffer);
```

### Case 3: Operation on an image with complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,  
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,  
    Ipp8u* pBuffer);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);  
IppStatus ippiFFTInv_PackToR_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

### Case 5: In-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

## DFTInitAlloc

Allocates memory and fills in context data needed for the image DFT functions to operate.

```
IppStatus ippiDFTInitAlloc_R_32s (IppiDFTSpec_R_32s** pDFTSpec, IppiSize  
    roiSize, int flag, IppHintAlgorithm hint);  
IppStatus ippiDFTInitAlloc_R_32f (IppiDFTSpec_R_32f** pDFTSpec, IppiSize  
    roiSize, int flag, IppHintAlgorithm hint);  
IppStatus ippiDFTInitAlloc_C_32fc (IppiDFTSpec_C_32fc** pDFTSpec,  
    IppiSize roiSize, int flag, IppHintAlgorithm hint);
```

## DFTFree

Deallocates memory used by the DFT context structure.

```
IppStatus ippiDFTFree_R_32s (IppiDFTSpec_R_32s* pDFTSpec);  
IppStatus ippiDFTFree_R_32f (IppiDFTSpec_R_32f* pDFTSpec);  
IppStatus ippiDFTFree_C_32fc (IppiDFTSpec_C_32fc* pDFTSpec);
```

## DFTGetBufSize

Determines the size of work buffer that can be used by the DFT functions to operate.

```
IppStatus ippiDFTGetBufSize_R_32s(IppiDFTSpec_R_32s* pDFTSpec, int*  
    pSize);
```

```
IppStatus ippiDFTGetBufSize_R_32f(IppiDFTSpec_R_32f* pDFTSpec, int*
    pSize);
IppStatus ippiDFTGetBufSize_C_32fc(IppiDFTSpec_C_32fc* pDFTSpec, int*
    pSize);
```

## DFTFwd

Applies forward discrete Fourier transform to an image.

### Case 1: Operation on an image with integer data

```
IppStatus ippiDFTFwd_RToPack_8u32s_C1RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s*
    pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_8u32s_C3RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s*
    pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_8u32s_C4RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s*
    pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_8u32s_AC4RSfs (const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s*
    pDFTSpec, int scaleFactor, Ipp8u* pBuffer);
```

### Case 2: Operation on an image with floating-point data

```
IppStatus ippiDFTFwd_RToPack_32f_C1R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_C3R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_C4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_AC4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

### Case 3: Operation on an image with complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
    Ipp8u* pBuffer);
```



#### Case 4: In-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTFwd_RToPack_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

#### Case 5: In-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

### DFTInv

Applies an inverse DFT to complex source data and stores results in a destination image.

#### Case 1: Operation on an image with integer data

```
IppStatus ippiDFTInv_PackToR_32s8u_C1RSfs (const Ipp32s* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32s8u_C3RSfs (const Ipp32s* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32s8u_C4RSfs (const Ipp32s* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32s8u_AC4RSfs (const Ipp32s* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

#### Case 2: Operation on an image with floating-point data

```
IppStatus ippiDFTInv_PackToR_32f_C1R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32f_C3R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32f_C4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

```
IppStatus ippiDFTInv_PackToR_32f_AC4R (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

**Case 3: Operation on an image with complex data**

```
IppStatus ippiDFTInv_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
    Ipp8u* pBuffer);
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiDFTInv_PackToR_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippiDFTInv_PackToR_32f_AC4IR(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

**Case 5: In-place operation on complex data**

```
IppStatus ippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

## MulPack

Multiplies two source images with data in packed format and stores the result in a destination image in packed format.

**Case 1: Not-in-place operation on integer data**

```
IppStatus ippiMulPack_16s_C1RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_32s_C1RSfs(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_16s_C3RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_32s_C3RSfs(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
```

```
IppStatus ippiMulPack_16s_C4RSfs(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_32s_C4RSfs(const Ipp32s* pSrc1, int src1Step, const
    Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_16s_AC4RSfs(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
IppStatus ippiMulPack_32s_AC4RSfs(const Ipp32s* pSrc1, int src1Step,
    const Ipp32s* pSrc2, int src2Step, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, int sFactor);
```

### **Case 2: Not-in-place operation on floating-point data**

```
IppStatus ippiMulPack_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulPack_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulPack_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
IppStatus ippiMulPack_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
```

### **Case 3: In-place operation on integer data**

```
IppStatus ippiMulPack_16s_C1IRSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_32s_C1IRSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_16s_C3IRSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_32s_C3IRSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_16s_C4IRSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_32s_C4IRSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
```

```

IppStatus ippiMulPack_16s_AC4IRSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);
IppStatus ippiMulPack_32s_AC4IRSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pSrcDst, int dstSrcStep, IppiSize roiSize, int sFactor);

```

#### Case 4: In-place operation on floating-point data

```

IppStatus ippiMulPack_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int dstSrcStep, IppiSize roiSize);
IppStatus ippiMulPack_32f_C3IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int dstSrcStep, IppiSize roiSize);
IppStatus ippiMulPack_32f_C4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int dstSrcStep, IppiSize roiSize);
IppStatus ippiMulPack_32f_AC4IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pSrcDst, int dstSrcStep, IppiSize roiSize);

```

### MulPackConj

Multiplies a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.

```

IppStatus ippiMulPackConj_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_C1IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_C3IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_C4IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
IppStatus ippiMulPackConj_32f_AC4IR(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

```

### Magnitude

Computes magnitude of elements of a complex data image.

```

IppStatus ippiMagnitude_32fc32f_C1R(const Ipp32fc* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiMagnitude_32fc32f_C3R(const Ipp32fc* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize);

```

```
IppStatus ippiMagnitude_16sc16s_C1RSfs(const Ipp16sc* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiMagnitude_16sc16s_C3RSfs(const Ipp16sc* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

## MagnitudePack

Computes magnitude of elements of an image in packed format.

```
IppStatus ippiMagnitudePack_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiMagnitudePack_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiMagnitudePack_16s_C1RSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
IppStatus ippiMagnitudePack_16s_C3RSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
IppStatus ippiMagnitudePack_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
IppStatus ippiMagnitudePack_32s_C3RSfs(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

## Phase

Computes the phase of elements of a complex data image.

```
IppStatus ippiPhase_32fc32f_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiPhase_32fc32f_C3R(const Ipp32fc* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiPhase_16sc16s_C1RSfs(const Ipp16sc* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiPhase_32sc32s_C1RSfs(const Ipp32sc* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiPhase_16sc16s_C3RSfs(const Ipp16sc* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
IppStatus ippiPhase_32sc32s_C3RSfs(const Ipp32sc* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

## PhasePack

Computes the phase of elements of an image in packed format.

```
IppStatus ippiPhasePack_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiPhasePack_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize dstRoiSize);
IppStatus ippiPhasePack_16s_C1RSfs(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

```
IppStatus ippiPhasePack_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,  
    Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);  
IppStatus ippiPhasePack_16s_C3RSfs(const Ipp16s* pSrc, int srcStep,  
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);  
IppStatus ippiPhasePack_32s_C3RSfs(const Ipp32s* pSrc, int srcStep,  
    Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

## PolarToCart

Converts an image in the polar coordinate form to Cartesian coordinate form.

```
IppStatus ippiPolarToCart_32f32fc_P2C1R(const Ipp32f* pSrcMagn, const  
    Ipp32f* pSrcPhase, int srcStep, Ipp32fc* pDst, int dstStep, IppiSize  
    roiSize);
```

## PackToCplxExtend

Converts an image in packed format to a complex data image.

```
IppStatus ippiPackToCplxExtend_32s32sc_C1R(const Ipp32s* pSrc, IppiSize  
    srcSize, int srcStep, Ipp32sc* pDst, int dstStep);  
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc, IppiSize  
    srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

# Windowing Functions

## WinBartlett, WinBartlettSep

Applies Bartlett window function to the image.

### Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_8u_C1R (const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiWinBartlett_32f_C1R (const Ipp32f* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiWinBartlettSep_8u_C1R (const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);  
IppStatus ippiWinBartlettSep_32f_C1R (const Ipp32f* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiWinBartlett_8u_C1IR (Ipp8u* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

```
IppStatus ippiWinBartlett_32f_C1IR (Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiWinBartlettSep_8u_C1IR (Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiWinBartlettSep_32f_C1IR (Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

## WinHamming, WinHammingSep

Applies Hamming window function to the image.

### Case 1: Not-in-place operation

```
IppStatus ippiWinHamming_8u_C1R (const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiWinHamming_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize);
IppStatus ippiWinHammingSep_8u_C1R (const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiWinHammingSep_32f_C1R (const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation

```
IppStatus ippiWinHamming_8u_C1IR (Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiWinHamming_32f_C1IR (Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiWinHammingSep_8u_C1IR (Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize);
IppStatus ippiWinHammingSep_32f_C1IR (Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

## Discrete Cosine Transforms

### DCTFwdInitAlloc

Allocates memory and fills in context data needed for the forward DCT function to operate.

```
IppStatus ippiDCTFwdInitAlloc_32f (IppiDCTFwdSpec_32f** pDCTSpec,
    IppiSize roiSize, IppHintAlgorithm hint);
```

### DCTInvInitAlloc

Allocates memory and fills in context data needed for the inverse DCT function to operate.

```
IppStatus ippIDCTInvInitAlloc_32f (IppIDCTInvSpec_32f** pDCTSpec,  
    IppiSize roiSize, IppHintAlgorithm hint);
```

### DCTFwdFree

Deallocates memory used by the forward DCT context structure.

```
IppStatus ippIDCTFwdFree_32f (IppIDCTFwdSpec_32f** pDCTSpec);
```

### DCTInvFree

Deallocates memory used by the inverse DCT context structure.

```
IppStatus ippIDCTInvFree_32f (IppIDCTInvSpec_32f** pDCTSpec);
```

### DCTFwdGetBufSize

Determines the size of work buffer can be used by the forward DCT function to operate.

```
IppStatus ippIDCTFwdGetBufSize_32f (IppIDCTFwdSpec_32f* pDCTSpec, int*  
    pSize);
```

### DCTInvGetBufSize

Determines the size of work buffer that can be used by the inverse DCT function to operate.

```
IppStatus ippIDCTInvGetBufSize_32f (IppIDCTInvSpec_32f* pDCTSpec, int*  
    pSize);
```

### DCTFwd

Applies a forward discrete cosine transform to an image.

```
IppStatus ippIDCTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pDst, int dstStep, const IppIDCTFwdSpec_32f* pDCTSpec, Ipp32f*  
    pBuffer);
```

```
IppStatus ippIDCTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pDst, int dstStep, const IppIDCTFwdSpec_32f* pDCTSpec, Ipp32f*  
    pBuffer);
```

```
IppStatus ippIDCTFwd_32f_C4R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pDst, int dstStep, const IppIDCTFwdSpec_32f* pDCTSpec, Ipp32f*  
    pBuffer);
```

```
IppStatus ippIDCTFwd_32f_AC4R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pDst, int dstStep, const IppIDCTFwdSpec_32f* pDCTSpec, Ipp32f*  
    pBuffer);
```



## DCTInv

Applies an inverse discrete cosine transform to an image.

```
IppStatus ippiDCTInv_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp32f*
    pBuffer);

IppStatus ippiDCTInv_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp32f*
    pBuffer);

IppStatus ippiDCTInv_32f_C4R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp32f*
    pBuffer);

IppStatus ippiDCTInv_32f_AC4R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp32f*
    pBuffer);
```

## DCT8x8Fwd

Performs a forward DCT on a 2D buffer of 8x8 size.

### Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Fwd_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Fwd_32f_C1(const Ipp32f* pSrc, Ipp32f* pDst);
```

### Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Fwd_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
    pDst);
IppStatus ippiDCT8x8Fwd_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s*
    pDst);
```

### Case 3: In-place operation

```
IppStatus ippiDCT8x8Fwd_16s_C1I(Ipp16s* pSrcDst );
IppStatus ippiDCT8x8Fwd_32f_C1I(Ipp32f* pSrcDst );
```

## DCT8x8Inv

Performs an inverse DCT on a 2D buffer of 8x8 size.

### Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_32f_C1(const Ipp32f* pSrc, Ipp32f* pDst);
```

### Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Inv_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, int
    dstStep);
IppStatus ippiDCT8x8Inv_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int
    dstStep);
```

**Case 3: In-place operation**

```
IppStatus ippiDCT8x8Inv_16s_C1I(Ipp16s* pSrcDst);  
IppStatus ippiDCT8x8Inv_32f_C1I(Ipp32f* pSrcDst);
```

**DCT8x8FwdLS**

Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp16s* pDst, Ipp16s addVal);
```

**DCT8x8InvLSClip**

Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst,  
    int dstStep, Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

**DCT8x8Inv\_2x2,  
DCT8x8Inv\_4x4**

Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);  
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);  
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);  
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

## Image Statistics Functions

### Sum

Computes the sum of image pixel values.

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiSum_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f* pSum);  
IppStatus ippiSum_16s_C1R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f* pSum);
```

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

#### Case 3: Operation on multi-channel integer data

```
IppStatus ippiSum_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3]);  
IppStatus ippiSum_16s_C3R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3]);  
IppStatus ippiSum_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3]);  
IppStatus ippiSum_16s_AC4R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3]);  
IppStatus ippiSum_8u_C4R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[4]);  
IppStatus ippiSum_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[4]);
```

#### Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiSum_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3], IppHintAlgorithm hint);  
IppStatus ippiSum_32f_AC4R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[3], IppHintAlgorithm hint);  
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f sum[4], IppHintAlgorithm hint);
```

### Integral

Transforms an image to the integral representation.

```
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*  
    pDst, int dstStep, IppiSize roiSize, Ipp32s val);  
IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*  
    pDst, int dstStep, IppiSize roiSize, Ipp32f val);
```

## SqrIntegral

Transforms an image to integral and integral of pixel squares representations.

```
IppStatus ippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize
    roiSize, Ipp32s val, Ipp32s valSqr);

IppStatus ippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize
    roiSize, Ipp32s val, Ipp64f valSqr);

IppStatus ippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize
    roiSize, Ipp32f val, Ipp64f valSqr);
```

## TiltedIntegral

Transforms an image to the tilted integral representation.

```
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, Ipp32s val);

IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f val);
```

## TiltedSqrIntegral

Transforms an image to tilted integral and tilted integral of pixel squares representations.

```
IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep,
    IppiSize roiSize, Ipp32s val, Ipp32s valSqr);

IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep,
    IppiSize roiSize, Ipp32s val, Ipp64f valSqr);

IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep,
    IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

## Mean

Computes the mean of image pixel values.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiMean_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pMean);

IppStatus ippiMean_16s_C1R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pMean);
```

**Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pMean, IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiMean_8u_C1MR(const Ipp8u* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);  
IppStatus ippiMean_8s_C1MR(const Ipp8s* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);  
IppStatus ippiMean_16u_C1MR(const Ipp16u* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);  
IppStatus ippiMean_32f_C1MR(const Ipp32f* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);
```

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiMean_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3]);  
IppStatus ippiMean_16s_C3R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3]);  
IppStatus ippiMean_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3]);  
IppStatus ippiMean_16s_AC4R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3]);  
IppStatus ippiMean_8u_C4R(const Ipp8u* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[4]);  
IppStatus ippiMean_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[4]);
```

**Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiMean_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3], IppHintAlgorithm hint);  
IppStatus ippiMean_32f_AC4R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[3], IppHintAlgorithm hint);  
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize  
    roiSize, Ipp64f mean[4], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiMean_8u_C3CMR(const Ipp8u* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);  
IppStatus ippiMean_8s_C3CMR(const Ipp8s* pSrc, int srcStep, const Ipp8u*  
    pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);  
IppStatus ippiMean_16u_C3CMR(const Ipp16u* pSrc, int srcStep, const  
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*  
    pMean);  
IppStatus ippiMean_32f_C3CMR(const Ipp32f* pSrc, int srcStep, const  
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*  
    pMean);
```

## Mean\_StdDev

Computes the mean and standard deviation of image pixel values.

### Case 1: Operation on one-channel data

```
IppStatus ippiMean_StdDev_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_8s_C1R(const Ipp8s* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_16u_C1R(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

### Case 2: Masked operation on one-channel data

```
IppStatus ippiMean_StdDev_8u_C1MR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f*
    pStdDev);
IppStatus ippiMean_StdDev_8s_C1MR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f*
    pStdDev);
IppStatus ippiMean_StdDev_16u_C1MR(const Ipp16u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f*
    pStdDev);
IppStatus ippiMean_StdDev_32f_C1MR(const Ipp32f* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f*
    pStdDev);
```

### Case 3: Operation on multi-channel data

```
IppStatus ippiMean_StdDev_8u_C3CR(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_8s_C3CR(const Ipp8s* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_16u_C3CR(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_32f_C3CR(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

### Case 4: Masked operation on multi-channel data

```
IppStatus ippiMean_StdDev_8u_C3CMR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean,
    Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_8s_C3CMR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean,
    Ipp64f* pStdDev);
IppStatus ippiMean_StdDev_16u_C3CMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pMean, Ipp64f* pStdDev);
```

```
IppStatus ippiMean_StdDev_32f_C3CMR(const Ipp32f* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pMean, Ipp64f* pStdDev);
```

## RectStdDev

Computes the standard deviation of the integral images.

```
IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const
    Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);

IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect, int scaleFactor);

IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
```

## TiltedRectStdDev

Computes the standard deviation of the tilted integral images.

```
IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);

IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect, int scaleFactor);

IppStatus ippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int
    srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect);
```

## HistogramRange

Computes the intensity histogram of an image.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiHistogramRange_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32s* pLevels, int nLevels);

IppStatus ippiHistogramRange_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32s* pLevels, int nLevels);
```

### Case 2: Operation on multi-channel integer data

```
IppStatus ippiHistogramRange_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

```

IppStatus ippiHistogramRange_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiHistogramRange_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiHistogramRange_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
IppStatus ippiHistogramRange_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
IppStatus ippiHistogramRange_16s_C4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], const Ipp32s* pLevels[4], int
    nLevels[4]);

```

### Case 3: Operation on one-channel floating-point data

```

IppStatus ippiHistogramRange_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32f* pLevels, int nLevels);

```

### Case 4: Operation on multi-channel floating-point data

```

IppStatus ippiHistogramRange_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
IppStatus ippiHistogramRange_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32f* pLevels[3], int
    nLevels[3]);
IppStatus ippiHistogramRange_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize roiSize, Ipp32s* pHist[4], const Ipp32f* pLevels[4],
    int nLevels[4]);

```

## HistogramEven

Computes the intensity histogram of an image using equal bins.

### Case 1: Operation on one-channel integer data

```

IppStatus ippiHistogramEven_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, Ipp32s* pLevels, int nLevels, Ipp32s
    lowerLevel, Ipp32s upperLevel);
IppStatus ippiHistogramEven_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, Ipp32s* pLevels, int nLevels, Ipp32s
    lowerLevel, Ipp32s upperLevel);

```

### Case 2: Operation on multi-channel integer data

```

IppStatus ippiHistogramEven_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3], int
    nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
IppStatus ippiHistogramEven_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3], int
    nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);

```



```
IppStatus ippiHistogramEven_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3], int
    nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
IppStatus ippiHistogramEven_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3], int
    nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
IppStatus ippiHistogramEven_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], Ipp32s* pLevels[4], int
    nLevels[4], Ipp32s lowerLevel[4], Ipp32s upperLevel[4]);
IppStatus ippiHistogramEven_16s_C4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], Ipp32s* pLevels[4], int
    nLevels[4], Ipp32s lowerLevel[4], Ipp32s upperLevel[4]);
```

## CountInRange

Computes the number of pixels within the given intensity range.

### Case 1: Operation on one-channel data

```
IppStatus ippiCountInRange_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, int* counts, Ipp8u lowerBound, Ipp8u upperBound);
IppStatus ippiCountInRange_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, int* counts, Ipp32f lowerBound, Ipp32f upperBound);
```

### Case 2: Operation on multi-channel data

```
IppStatus ippiCountInRange_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, int counts[3], Ipp8u lowerBound[3], Ipp8u
    upperBound[3]);
IppStatus ippiCountInRange_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, int counts[3], Ipp32f lowerBound[3], Ipp32f
    upperBound[3]);
IppStatus ippiCountInRange_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, int counts[3], Ipp8u lowerBound[3], Ipp8u
    upperBound[3]);
IppStatus ippiCountInRange_32f_AC4(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, int counts[3], Ipp32f lowerBound[3], Ipp32f
    upperBound[3]);
```

## Min

Computes the minimum of image pixel values.

### Case 1: Operation on one-channel data

```
IppStatus ippiMin_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u* pMin);
IppStatus ippiMin_16s_C1R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s* pMin);
IppStatus ippiMin_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f* pMin);
```

**Case 2: Operation on multi-channel data**

```

IppStatus ippiMin_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u min[3]);
IppStatus ippiMin_16s_C3R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[3]);
IppStatus ippiMin_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[3]);
IppStatus ippiMin_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u min[3]);
IppStatus ippiMin_16s_AC4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[3]);
IppStatus ippiMin_32f_AC4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[3]);
IppStatus ippiMin_8u_C4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u min[4]);
IppStatus ippiMin_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[4]);
IppStatus ippiMin_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[4]);

```

**MinIndx**

Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.

**Case 1: Operation on one-channel data**

```

IppStatus ippiMinIndx_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u* pMin, int* pIndexX, int* pIndexY);
IppStatus ippiMinIndx_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s* pMin, int* pIndexX, int* pIndexY);
IppStatus ippiMinIndx_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMin, int* pIndexX, int* pIndexY);

```

**Case 2: Operation on multi-channel data**

```

IppStatus ippiMinIndx_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f min[3], int indexX[3], int indexY[3]);
IppStatus ippiMinIndx_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u min[4], int indexX[4], int indexY[4]);

```

```
IppStatus ippiMinIndx_16s_C4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s min[4], int indexX[4], int indexY[4]);
IppStatus ippiMinIndx_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f min[4], int indexX[4], int indexY[4]);
```

## Max

Computes the maximum of image pixel values.

### Case 1: Operation on one-channel data

```
IppStatus ippiMax_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u* pMax);
IppStatus ippiMax_16s_C1R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s* pMax);
IppStatus ippiMax_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f* pMax);
```

### Case 2: Operation on multi-channel data

```
IppStatus ippiMax_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u max[3]);
IppStatus ippiMax_16s_C3R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s max[3]);
IppStatus ippiMax_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f max[3]);
IppStatus ippiMax_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u max[3]);
IppStatus ippiMax_16s_AC4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s max[3]);
IppStatus ippiMax_32f_AC4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f max[3]);
IppStatus ippiMax_8u_C4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u max[4]);
IppStatus ippiMax_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s max[4]);
IppStatus ippiMax_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f max[4]);
```

## MaxIndx

Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.

### Case 1: Operation on one-channel data

```
IppStatus ippiMaxIndx_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u* pMax, int* pIndexX, int* pIndexY);
IppStatus ippiMaxIndx_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s* pMax, int* pIndexX, int* pIndexY);
IppStatus ippiMaxIndx_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMax, int* pIndexX, int* pIndexY);
```

**Case 2: Operation on multi-channel data**

```

IppStatus ippiMaxIndx_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f max[3], int indexX[3], int indexY[3]);
IppStatus ippiMaxIndx_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u max[4], int indexX[4], int indexY[4]);
IppStatus ippiMaxIndx_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s max[4], int indexX[4], int indexY[4]);
IppStatus ippiMaxIndx_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f max[4], int indexX[4], int indexY[4]);

```

**MinMax**

Computes the minimum and maximum of image pixel values.

**Case 1: Operation on one-channel data**

```

IppStatus ippiMinMax_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u* pMin, Ipp8u* pMax);
IppStatus ippiMinMax_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp16s* pMin, Ipp16s* pMax);
IppStatus ippiMinMax_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMin, Ipp32f* pMax);

```

**Case 2: Operation on multi-channel data**

```

IppStatus ippiMinMax_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp8u min[3], Ipp8u max[3]);
IppStatus ippiMinMax_16s_C3R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[3], Ipp16s max[3]);
IppStatus ippiMinMax_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[3], Ipp32f max[3]);
IppStatus ippiMinMax_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u min[3], Ipp8u max[3]);
IppStatus ippiMinMax_16s_AC4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[3], Ipp16s max[3]);
IppStatus ippiMinMax_32f_AC4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[3], Ipp32f max[3]);
IppStatus ippiMinMax_8u_C4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp8u min[4], Ipp8u max[4]);

```

```
IppStatus ippiMinMax_16s_C4R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp16s min[4], Ipp16s max[4]);
IppStatus ippiMinMax_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f min[4], Ipp32f max[4]);
```

## MinMaxIndx

Calculates minimum and maximum pixel values and their indexes in selected image rectangle.

### Case 1: Operation on one-channel data

```
IppStatus ippiMinMaxIndx_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_8s_C1R(const Ipp8s* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_16u_C1R(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
    pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
    pMinIndex, IppiPoint* pMaxIndex);
```

### Case 2: Masked operation on one-channel data

```
IppStatus ippiMinMaxIndx_8u_C1MR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_8s_C1MR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_16u_C1MR(const Ipp16u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_32f_C1MR(const Ipp32f* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

### Case 3: Operation on multi-channel data

```
IppStatus ippiMinMaxIndx_8u_C3CR(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
    pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_8s_C3CR(const Ipp8s* pSrc, int srcStep, IppiSize
    roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
    pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_16u_C3CR(const Ipp16u* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
IppStatus ippiMinMaxIndx_32f_C3CR(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippMinMaxIndx_8u_C3CMR(const Ipp8u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);

IppStatus ippMinMaxIndx_8s_C3CMR(const Ipp8s* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);

IppStatus ippMinMaxIndx_16u_C3CMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);

IppStatus ippMinMaxIndx_32f_C3CMR(const Ipp32f* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);
```

## Image Moments

### MomentInitAlloc

Allocates memory and initializes the moment context structure.

```
IppStatus ippMomentInitAlloc_64f(IppiMomentState_64f** pState,
    IppHintAlgorithm hint);

IppStatus ippMomentInitAlloc_64s(IppiMomentState_64s** pState,
    IppHintAlgorithm hint);
```

### MomentFree

Frees memory allocated by the function ippMomentInitAlloc.

```
IppStatus ippMomentFree_64f(IppiMomentState_64f* pState);

IppStatus ippMomentFree_64s(IppiMomentState_64s* pState);
```

### MomentGetStateSize

Computes the size of the external buffer for the moment context structure.

```
IppStatus ippMomentGetStateSize_64s(IppHintAlgorithm hint, int* pSize);
```

### MomentInit

Initializes the moment context structure.

```
IppStatus ippMomentInit_64s(IppiMomentState_64s* pState,
    IppHintAlgorithm hint);
```

## Moments

Computes all image moments of order 0 to 3 and Hu moment invariants.

### Case 1: Computation of floating-point results

```
IppStatus ippiMoments64f_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, IppiMomentState_64f* pState);
IppStatus ippiMoments64f_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64f* pState);
IppStatus ippiMoments64f_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, IppiMomentState_64f* pState);
IppStatus ippiMoments64f_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64f* pState);
IppStatus ippiMoments64f_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, IppiMomentState_64f* pState);
IppStatus ippiMoments64f_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64f* pState);
```

### Case 2: Computation of integer results

```
IppStatus ippiMoments64s_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64s* pState);
IppStatus ippiMoments64s_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64s* pState);
IppStatus ippiMoments64s_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64s* pState);
```

## GetSpatialMoment

Retrieves image spatial moment of the specified order, computed by `ippiMoments`.

```
IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f*
    pValue);
IppStatus ippiGetSpatialMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64s*
    pValue, int scaleFactor);
```

## GetNormalizedSpatialMoment

Retrieves the normalized value of the image spatial moment computed by `ippiMoments`.

```
IppStatus ippiGetNormalizedSpatialMoment_64f(IppiMomentState_64f*
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64f* pValue);
IppStatus ippiGetNormalizedSpatialMoment_64s(IppiMomentState_64s*
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64s* pValue, int scaleFactor);
```

## GetCentralMoment

Retrieves image central moment computed by `ippiMoments`.

```
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, Ipp64f* pValue);

IppStatus ippiGetCentralMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, Ipp64s* pValue, int scaleFactor);
```

## GetNormalizedCentralMoment

Retrieves the normalized value of the image central moment computed by `ippiMoments`.

```
IppStatus ippiGetNormalizedCentralMoment_64f(IppiMomentState_64f*
    pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);

IppStatus ippiGetNormalizedCentralMoment_64s(IppiMomentState_64s*
    pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue, int
    scaleFactor);
```

## GetHuMoments

Retrieves image Hu moment invariants computed by `ippiMoments` function.

```
IppStatus ippiGetHuMoments_64f(IppiMomentState_64f* pState, int
    nChannel, IppiHuMoment_64f* pHm);

IppStatus ippiGetHuMoments_64s(IppiMomentState_64s* pState, int
    nChannel, IppiHuMoment_64s* pHm, int scaleFactor);
```

## Image Norms

### Norm\_Inf

Computes the infinity norm of image pixel values.

#### Case 1: Operation on one-channel data

```
IppStatus ippiNorm_Inf_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pValue);

IppStatus ippiNorm_Inf_16s_C1R(const Ipp16s* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pValue);

IppStatus ippiNorm_Inf_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pValue);

IppStatus ippiNorm_Inf_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp64f* pValue);
```

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiNorm_Inf_8u_C1MR(const Ipp8u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```



```
ippiStatus ippiNorm_Inf_8s_C1MR(const Ipp8s* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);  
ippiStatus ippiNorm_Inf_16u_C1MR(const Ipp16u* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);  
ippiStatus ippiNorm_Inf_32f_C1MR(const Ipp32f* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

**Case 3: Operation on multi-channel data**

```
ippiStatus ippiNorm_Inf_8u_C3R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_16s_C3R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_32f_C3R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_8u_AC4R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_16s_AC4R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_32f_AC4R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
ippiStatus ippiNorm_Inf_8u_C4R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);  
ippiStatus ippiNorm_Inf_16s_C4R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);  
ippiStatus ippiNorm_Inf_32f_AC4R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);
```

**Case 4: Masked operation on multi-channel data**

```
ippiStatus ippiNorm_Inf_8u_C3CMR(const Ipp8u* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pNorm);  
ippiStatus ippiNorm_Inf_8s_C3CMR(const Ipp8s* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pNorm);  
ippiStatus ippiNorm_Inf_16u_C3CMR(const Ipp16u* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pNorm);  
ippiStatus ippiNorm_Inf_32f_C3CMR(const Ipp32f* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pNorm);
```

**Norm\_L1**

Computes the L1- norm of image pixel values.

**Case 1: Operation on one-channel integer data**

```
ippiStatus ippiNorm_L1_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue);
```

```
IppStatus ippiNorm_L1_16s_C1R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue);
```

### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L1_8u_C1MR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNorm_L1_8s_C1MR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNorm_L1_16u_C1MR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNorm_L1_32f_C1MR(const Ipp32f* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L1_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNorm_L1_16s_C3R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNorm_L1_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNorm_L1_16s_AC4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNorm_L1_8u_C4R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNorm_L1_16s_C4R(const Ipp16s* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
IppStatus ippiNorm_L1_32f_AC4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L1_8u_C3CMR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pNorm);
IppStatus ippiNorm_L1_8s_C3CMR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pNorm);
IppStatus ippiNorm_L1_16u_C3CMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

```
IppStatus ippiNorm_L1_32f_C3CMR(const Ipp32f* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pNorm);
```

## Norm\_L2

Computes the L2- norm of image pixel values.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue);  
IppStatus ippiNorm_L2_16s_C1R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue);
```

### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L2_8u_C1MR(const Ipp8u* pSrc, int srcStep, const  
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);  
IppStatus ippiNorm_L2_8s_C1MR(const Ipp8s* pSrc, int srcStep, const  
    Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);  
IppStatus ippiNorm_L2_16u_C1MR(const Ipp16u* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);  
IppStatus ippiNorm_L2_32f_C1MR(const Ipp32f* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L2_8u_C3R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
IppStatus ippiNorm_L2_16s_C3R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
IppStatus ippiNorm_L2_8u_AC4R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
IppStatus ippiNorm_L2_16s_AC4R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);  
IppStatus ippiNorm_L2_8u_C4R(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);  
IppStatus ippiNorm_L2_16s_C4R(const Ipp16s* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);
```

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C3R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);  
IppStatus ippiNorm_L2_32f_AC4R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);  
IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```

IppStatus ippiNorm_L2_8u_C3CMR(const Ipp8u* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pNorm);
IppStatus ippiNorm_L2_8s_C3CMR(const Ipp8s* pSrc, int srcStep, const
    Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f*
    pNorm);
IppStatus ippiNorm_L2_16u_C3CMR(const Ipp16u* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
IppStatus ippiNorm_L2_32f_C3CMR(const Ipp32f* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);

```

**NormDiff\_Inf**

Computes the infinity norm of differences between pixel values of two images.

**Case 1: Operation on one-channel data**

```

IppStatus ippiNormDiff_Inf_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormDiff_Inf_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormDiff_Inf_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);

```

**Case 2: Masked operation on one-channel data**

```

IppStatus ippiNormDiff_Inf_8u_C1MR(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_8s_C1MR(const Ipp8s* pSrc1, int src1Step,
    const Ipp8s* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_16u_C1MR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);

```

**Case 3: Operation on multi-channel data**

```

IppStatus ippiNormDiff_Inf_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_Inf_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_Inf_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);

```

```
IppStatus ippiNormDiff_Inf_8u_AC4R(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_Inf_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_Inf_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_Inf_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormDiff_Inf_16s_C4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
IppStatus ippiNormDiff_Inf_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
```

#### **Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNormDiff_Inf_8u_C3CMR(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_8s_C3CMR(const Ipp8s* pSrc1, int src1Step,
    const Ipp8s* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormDiff_Inf_32f_C3CMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

## **NormDiff\_L1**

Computes the L1- norm of differences between pixel values of two images.

#### **Case 1: Operation on one-channel integer data**

```
IppStatus ippiNormDiff_L1_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormDiff_L1_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

#### **Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

#### **Case 3: Masked operation on one-channel data**

```
IppStatus ippiNormDiff_L1_8u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

```
IppStatus ippiNormDiff_L1_8s_C1MR(const Ipp8s* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_L1_16u_C1MR(const Ipp8u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_L1_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

#### **Case 4: Operation on multi-channel integer data**

```
IppStatus ippiNormDiff_L1_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_L1_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_L1_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_L1_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_L1_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormDiff_L1_16s_C4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
```

#### **Case 5: Operation on multi-channel floating-point data**

```
IppStatus ippiNormDiff_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
IppStatus ippiNormDiff_L1_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

#### **Case 6: Masked operation on multi-channel data**

```
IppStatus ippiNormDiff_L1_8u_C3CMR(const Ipp8u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormDiff_L1_8s_C3CMR(const Ipp8s* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormDiff_L1_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

```
IppStatus ippiNormDiff_L1_32f_C3CMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

## NormDiff\_L2

Computes the L2- norm of differences between pixel values of two images.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L2_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormDiff_L2_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L2_8u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_L2_8s_C1MR(const Ipp8s* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_L2_16u_C1MR(const Ipp16u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormDiff_L2_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L2_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_L2_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_L2_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormDiff_L2_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormDiff_L2_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormDiff_L2_16s_C4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
```

**Case 5: Operation on multi-channel floating-point data**

```

IppStatus ippiNormDiff_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormDiff_L2_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);

```

**Case 6: Masked operation on multi-channel data**

```

IppStatus ippiNormDiff_L2_8u_C3CMR(const Ipp8u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormDiff_L2_8s_C3CMR(const Ipp8s* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormDiff_L2_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormDiff_L2_32f_C3CMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);

```

**NormRel\_Inf**

Computes the relative error for the infinity norm of differences between pixel values of two images.

**Case 1: Operation on one-channel data**

```

IppStatus ippiNormRel_Inf_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormRel_Inf_16s_C1R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormRel_Inf_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);

```

**Case 2: Masked operation on one-channel data**

```

IppStatus ippiNormRel_Inf_8u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_Inf_8s_C1MR(const Ipp8s* pSrc1, int src1Step, const
    Ipp8s* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_Inf_16u_C1MR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);

```



```
IppStatus ippiNormRel_Inf_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

### **Case 3: Operation on multi-channel data**

```
IppStatus ippiNormRel_Inf_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_Inf_16s_C3R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_Inf_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_Inf_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_Inf_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_Inf_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_Inf_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormRel_Inf_16s_C4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
IppStatus ippiNormRel_Inf_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[4]);
```

### **Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNormRel_Inf_8u_C3CMR(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_Inf_8s_C3CMR(const Ipp8s* pSrc1, int src1Step,
    const Ipp8s* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_Inf_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp16u* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_Inf_32f_C3CMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

## NormRel\_L1

Computes the relative error for the L1 norm of differences between pixel values of two images.

### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L1_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormRel_L1_16s_C1R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L1_8u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L1_8s_C1MR>(const Ipp8s* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L1_16u_C1MR(const Ipp16u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L1_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L1_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_L1_16s_C3R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_L1_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_L1_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_L1_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormRel_L1_16s_C4R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormRel_L1_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

**Case 6: Masked operation on multi-channel data**

```
IppStatus ippiNormRel_L1_8u_C3CMR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormRel_L1_8s_C3CMR(const Ipp8s* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormRel_L1_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
IppStatus ippiNormRel_L1_32f_C3CMR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
```

## NormRel\_L2

Computes the relative error for the L2 norm of differences between pixel values of two images.

**Case 1: Operation on one-channel integer data**

```
IppStatus ippiNormRel_L2_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
IppStatus ippiNormRel_L2_16s_C1R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

**Case 2: Operation on one-channel floating-point data**

```
IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

**Case 3: Masked operation on one-channel data**

```
IppStatus ippiNormRel_L2_8u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_8s_C1MR(const Ipp8s* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_16u_C1MR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_32f_C1MR(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

**Case 4: Operation on multi-channel integer data**

```
IppStatus ippiNormRel_L2_8u_C3R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

```

IppStatus ippiNormRel_L2_16s_C3R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_L2_8u_AC4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
IppStatus ippiNormRel_L2_16s_AC4R(const Ipp16s* pSrc1, int src1Step,
    const Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f
    value[3]);
IppStatus ippiNormRel_L2_8u_C4R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
IppStatus ippiNormRel_L2_16s_C4R(const Ipp16s* pSrc1, int src1Step, const
    Ipp16s* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

```

#### Case 5: Operation on multi-channel floating-point data

```

IppStatus ippiNormRel_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormRel_L2_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);

```

#### Case 6: Masked operation on multi-channel data

```

IppStatus ippiNormRel_L2_8u_C3CMR(const Ipp8u* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_8s_C3CMR(const Ipp8s* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_16u_C3CMR(const Ipp16u* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
IppStatus ippiNormRel_L2_32f_C3CM(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);

```

## Image Quality Index

### QualityIndex

Computes the universal image quality index.

#### Case 1: Operation on one-channel data

```

IppStatus ippiQualityIndex_8u32f_C1R(const Ipp8u32f* pSrc1, int src1Step,
    const Ipp8u32f* pSrc2, int src2Step, IppiSize roiSize, Ipp32f*
    pQualityIndex);

```

```
IppStatus ippiQualityIndex_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp32f*
    pQualityIndex);
```

**Case 2: Operation on multi-channel data**

```
IppStatus ippiQualityIndex_8u32f_C3R(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp32f
    qualityIndex[3]);
```

```
IppStatus ippiQualityIndex_32f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp32f
    qualityIndex[3]);
```

```
IppStatus ippiQualityIndex_8u32f_AC4R(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, IppiSize roiSize, Ipp32f
    qualityIndex[3]);
```

```
IppStatus ippiQualityIndex_32f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp32f
    qualityIndex[3]);
```

## Image Proximity Measures

### SqrDistanceFull\_Norm

Computes normalized Euclidean distance between an image and a template.

**Case 1: Operation with integer output**

```
IppStatus ippiSqrDistanceFull_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

```
IppStatus ippiSqrDistanceFull_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

```
IppStatus ippiSqrDistanceFull_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

```
IppStatus ippiSqrDistanceFull_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

**Case 2: Operation on data with floating-point output**

```
IppStatus ippiSqrDistanceFull_Norm_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```
IppStatus ippiSqrDistanceFull_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```
IppStatus ippiSqrDistanceFull_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```

IppStatus ippiSqrDistanceFull_Norm_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceFull_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```

## SqrDistanceSame\_Norm

Computes normalized Euclidean distance between an image and a template.

### Case 1: Operation with integer output

```

IppStatus ippiSqrDistanceSame_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiSqrDistanceSame_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiSqrDistanceSame_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiSqrDistanceSame_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);

```

**Case 2: Operation on data with floating-point output**

```
IppStatus ippiSqrDistanceSame_Norm_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceSame_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

**SqrDistanceValid\_Norm**

Computes normalized Euclidean distance between an image and a template.

**Case 1: Operation with integer output**

```
IppStatus ippiSqrDistanceValid_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

```

IppStatus ippiSqrDistanceValid_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiSqrDistanceValid_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiSqrDistanceValid_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);

```

## Case 2: Operation on data with floating-point output

```

IppStatus ippiSqrDistanceValid_Norm_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiSqrDistanceValid_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```



## CrossCorrFull\_Norm

Computes normalized cross-correlation between an image and a template.

### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_Norm_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```

IppStatus ippiCrossCorrFull_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```

## CrossCorrSame\_Norm

Computes normalized cross-correlation between an image and a template.

### Case 1: Operation with integer output

```

IppStatus ippiCrossCorrSame_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrSame_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrSame_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrSame_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);

```

### Case 2: Operation on data with floating-point output

```

IppStatus ippiCrossCorrSame_Norm_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep, IppiSize
    tplRoiSize, Ipp32f* pDst, int dstStep);

```

```
IppStatus ippiCrossCorrSame_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

## CrossCorrValid\_Norm

Computes normalized cross-correlation between an image and a template.

### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_Norm_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_Norm_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_Norm_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_Norm_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_Norm_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```

IppStatus ippiCrossCorrValid_Norm_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_Norm_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```

## CrossCorrFull\_NormLevel

Computes normalized correlation coefficient between an image and a template.

### Case 1: Operation with integer output

```

IppStatus ippiCrossCorrFull_NormLevel_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_NormLevel_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_NormLevel_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrFull_NormLevel_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);

```

### Case 2: Operation on data with floating-point output

```

IppStatus ippiCrossCorrFull_NormLevel_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```

```
IppStatus ippiCrossCorrFull_NormLevel_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrFull_NormLevel_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

## CrossCorrSame\_NormLevel

Computes normalized correlation coefficient between an image and a template.

### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_NormLevel_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrSame_NormLevel_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrSame_NormLevel_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

---

```
IppStatus ippiCrossCorrSame_NormLevel_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

### **Case 2: Operation on data with floating-point output**

```
IppStatus ippiCrossCorrSame_NormLevel_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrSame_NormLevel_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

## CrossCorrValid\_NormLevel

Computes normalized cross-correlation between an image and a template.

### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_NormLevel_8u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_NormLevel_8u_C3RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_NormLevel_8u_C4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
IppStatus ippiCrossCorrValid_NormLevel_8u_AC4RSfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_NormLevel_32f_C1R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8s32f_C1R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_32f_C3R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8u32f_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8s32f_C3R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_32f_C4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8u32f_C4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8s32f_C4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_32f_AC4R(const Ipp32f* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp32f* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

```

IppStatus ippiCrossCorrValid_NormLevel_8u32f_AC4R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
IppStatus ippiCrossCorrValid_NormLevel_8s32f_AC4R(const Ipp8s* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp8s* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);

```

## Image Geometric Transforms

### Resize

Changes an image size.

#### Case 1: Operation on pixel-order data

```

IppStatus ippiResize_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_16u_C1R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_16u_C3R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_16u_C4R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
IppStatus ippiResize_16u_AC4R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);

```



```
IppStatus ippiResize_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
```

### **Case 2: Operation on planar-order data**

```
IppStatus ippiResize_8u_P3R(const Ipp8u* const pSrc[3], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int dstStep,
    IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

```
IppStatus ippiResize_16u_P3R(const Ipp16u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp16u* const pDst[3], int
    dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

```
IppStatus ippiResize_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[3], int
    dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

```
IppStatus ippiResize_8u_P4R(const Ipp8u* const pSrc[4], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int dstStep,
    IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

```
IppStatus ippiResize_16u_P4R(const Ipp16u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp16u* const pDst[4], int
    dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

```
IppStatus ippiResize_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[4], int
    dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

## **ResizeCenter**

Changes an image size and shifts the destination image relative to the given point.

### **Case 1: Operation on pixel-order data**

```
IppStatus ippiResizeCenter_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
```

```
IppStatus ippiResizeCenter_16u_C1R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
```

```
IppStatus ippiResizeCenter_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
```

---

```

IppStatus ippiResizeCenter_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_16u_C3R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_16u_C4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_16u_AC4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiResizeCenter_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, double xCenter, double
    yCenter, int interpolation);

```

### Case 2: Operation on planar-order data

```

IppStatus ippiResizeCenter_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor,
    double xCenter, double yCenter, int interpolation);
IppStatus ippiResizeCenter_16u_P3R(const Ipp16u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[3], int dstStep, IppiSize dstRoiSize, double xFactor, double
    yFactor, double xCenter, double yCenter, int interpolation);

```

```
IppStatus ippiResizeCenter_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiSize dstRoiSize, double xFactor, double
    yFactor, double xCenter, double yCenter, int interpolation);
IppStatus ippiResizeCenter_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor,
    double xCenter, double yCenter, int interpolation);
IppStatus ippiResizeCenter_16u_P4R(const Ipp16u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[4], int dstStep, IppiSize dstRoiSize, double xFactor, double
    yFactor, double xCenter, double yCenter, int interpolation);
IppStatus ippiResizeCenter_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiSize dstRoiSize, double xFactor, double
    yFactor, double xCenter, double yCenter, int interpolation);
```

## ResizeSqrPixel

Changes an image size using different interpolation algorithm.

### Case 1: Operation on pixel-order data

```
IppStatus ippiResizeSqrPixel_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiRect
    dstRoi, double xFactor, double yFactor, double xShift, double yShift,
    int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiRect
    dstRoi, double xFactor, double yFactor, double xShift, double yShift,
    int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiRect
    dstRoi, double xFactor, double yFactor, double xShift, double yShift,
    int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiRect
    dstRoi, double xFactor, double yFactor, double xShift, double yShift,
    int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_16u_C1R(const Ipp16u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp16u* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_16u_C3R(const Ipp16u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp16u* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
```

```
IppStatus ippiResizeSqrPixel_16u_C4R(const Ipp16u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp16u* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_16u_AC4R(const Ipp16u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp16u* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp32f* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp32f* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp32f* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp32f* pDst, int dstStep,
    IppiRect dstRoi, double xFactor, double yFactor, double xShift,
    double yShift, int interpolation, Ipp8u* pBuffer);
```

**Case 2: Operation on planar-order data**

```
IppStatus ippiResizeSqrPixel_8u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp8u* const pDst[3], int
    dstStep, IppiRect dstRoi, double xFactor, double yFactor, double
    xShift, double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_16u_P3R(const Ipp16u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp16u* const
    pDst[3], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
IppStatus ippiResizeSqrPixel_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
IppStatus ippiResizeSqrPixel_8u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp8u* const pDst[4], int
    dstStep, IppiRect dstRoi, double xFactor, double yFactor, double
    xShift, double yShift, int interpolation, Ipp8u* pBuffer);
IppStatus ippiResizeSqrPixel_16u_P4R(const Ipp16u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp16u* const
    pDst[4], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
```

```
IppStatus ippiResizeSqrPixel_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
```

## ResizeSqrPixelGetBufSize

Computes the size of the external buffer for image resizing.

```
IppStatus ippiResizeSqrPixelGetBufSize(IppiSize dstSize, int nChannel,
    int interpolation, int* pBufferSize);
```

## GetResizeFract

Recalculates resize factors for a tiled image.

```
IppStatus ippiGetResizeFract(IppiSize srcSize, IppiRect srcRoi, double
    xFactor, double yFactor, double* xFr, double* yFr, int
    interpolation);
```

## ResizeShift

Changes the size of an image tile.

### Case 1: Operation on pixel-order data

```
IppStatus ippiResizeShift_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_16u_C1R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_16u_C3R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
```

```
IppStatus ippiResizeShift_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_16u_C4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_16u_AC4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
IppStatus ippiResizeShift_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, double xFr, double yFr, double xShift, double yShift, int
    interpolation);
```

**Case 2: Operation on planar-order data**

```
IppStatus ippiResizeShift_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
IppStatus ippiResizeShift_16u_P3R(const Ipp16u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[3], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
IppStatus ippiResizeShift_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
IppStatus ippiResizeShift_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
IppStatus ippiResizeShift_16u_P4R(const Ipp16u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[4], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
```

```
IppStatus ippiResizeShift_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
```

## ResizeYUV422

Changes a size of two-channel image.

```
IppStatus ippiResizeYUV422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
```

## Mirror

Mirrors an image about a horizontal or vertical axis, or both.

### Case 1: Not-in-place operation

```
IppStatus ippiMirror_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_C1R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_C3R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_C4R(const Ipp32s* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_AC4R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_AC4R(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
```

### Case 2: In-place operation

```
IppStatus ippiMirror_8u_C1IR(const Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C1IR(const Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
```

---

```

IppStatus ippiMirror_32s_C1IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_C3IR(const Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C3IR(const Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_C3IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_C4IR(const Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_C4IR(const Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_C4IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_8u_AC4IR(const Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_16u_AC4IR(const Ipp16u* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
IppStatus ippiMirror_32s_AC4IR(const Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);

```

## Remap

Applies the look-up coordinate mapping to pixels of a source image.

### Case 1: Operation on pixel-order data

```

IppStatus ippiRemap_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);

```



```
IppStatus ippiRemap_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
IppStatus ippiRemap_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep, const
    Ipp32f* pyMap, int yMapStep, Ipp32f* pDst, int dstStep, IppiSize
    dstRoiSize, int interpolation);
```

### **Case 2: Operation on planar-order data**

```
IppStatus ippiRemap_8u_P3R(const Ipp8u* const pSrc[3], IppiSize srcSize,
    int srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep,
    const Ipp32f* pyMap, int yMapStep, Ipp8u* const pDst[3], int dstStep,
    IppiSize dstRoiSize, int interpolation);
IppStatus ippiRemap_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const Ipp32f* pxMap, int
    xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp32f* const pDst[3],
    int dstStep, IppiSize dstRoiSize, int interpolation);
IppStatus ippiRemap_8u_P4R(const Ipp8u* const pSrc[4], IppiSize srcSize,
    int srcStep, IppiRect srcROI, const Ipp32f* pxMap, int xMapStep,
    const Ipp32f* pyMap, int yMapStep, Ipp8u* const pDst[4], int dstStep,
    IppiSize dstRoiSize, int interpolation);
IppStatus ippiRemap_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const Ipp32f* pxMap, int
    xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp32f* const pDst[4],
    int dstStep, IppiSize dstRoiSize, int interpolation);
```

## **Rotate**

Rotates an image around the origin (0,0) and shifts it.

### **Case 1: Operation on pixel-order data**

```
IppStatus ippiRotate_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippiRotate_16u_C1R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippiRotate_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippiRotate_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
```

```
IppStatus ippRotate_16u_C3R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_16u_C4R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_16u_AC4R(const Ipp16u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
IppStatus ippRotate_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double angle, double xShift, double yShift, int interpolation);
```

**Case 2: Operation on planar-order data**

```
IppStatus ippRotate_8u_P3R(const Ipp8u* const pSrc[3], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int dstStep,
    IppiRect dstROI, double angle, double xShift, double yShift, int
    interpolation);
IppStatus ippRotate_16u_P3R(const Ipp16u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp16u* const pDst[3], int
    dstStep, IppiRect dstROI, double angle, double xShift, double yShift,
    int interpolation);
IppStatus ippRotate_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[3], int
    dstStep, IppiRect dstROI, double angle, double xShift, double yShift,
    int interpolation);
IppStatus ippRotate_8u_P4R(const Ipp8u* const pSrc[4], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int dstStep,
    IppiRect dstROI, double angle, double xShift, double yShift, int
    interpolation);
IppStatus ippRotate_16u_P4R(const Ipp16u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp16u* const pDst[4], int
    dstStep, IppiRect dstROI, double angle, double xShift, double yShift,
    int interpolation);
```

```
IppStatus ippiRotate_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[4], int
    dstStep, IppiRect dstROI, double angle, double xShift, double yShift,
    int interpolation);
```

## GetRotateShift

Computes shift values for rotation of an image around the specified center.

```
IppStatus ippiGetRotateShift(double xCenter, double yCenter, double
    angle, double* xShift, double* yShift)
```

## AddRotateShift

Computes shift values for rotating an image around the specified center with specified shifts.

```
IppStatus ippiAddRotateShift(double xCenter, double yCenter, double
    angle, double* xShift, double* yShift)
```

## GetRotateQuad

Computes vertex coordinates of the quadrangle, to which the source ROI would be mapped by the `ippiRotate` function.

```
IppStatus ippiGetRotateQuad(IppiRect srcRoi, double quad[4][2], double
    angle, double xShift, double yShift);
```

## GetRotateBound

Computes the bounding rectangle for the source ROI transformed by the `ippiRotate` function.

```
IppStatus ippiGetRotateBound(IppiRect srcRoi, double bound[2][2], double
    angle, double xShift, double yShift);
```

## RotateCenter

Rotates an image about an arbitrary center.

### Case 1: Operation on pixel-order data

```
IppStatus ippiRotateCenter_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_16u_C1R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
```

```
IppStatus ippiRotateCenter_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_16u_C3R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_16u_C4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_16u_AC4R(const Ipp16u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp16u* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
IppStatus ippiRotateCenter_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, double angle, double xCenter, double yCenter, int
    interpolation);
```

**Case 2: Operation on planar-order data**

```
IppStatus ippiRotateCenter_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiRect dstROI, double angle, double xCenter, double
    yCenter, int interpolation);
```

```
IppStatus ippiRotateCenter_16u_P3R(const Ipp16u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[3], int dstStep, IppiRect dstROI, double angle, double xCenter,
    double yCenter, int interpolation);
IppStatus ippiRotateCenter_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstROI, double angle, double xCenter,
    double yCenter, int interpolation);
IppStatus ippiRotateCenter_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiRect dstROI, double angle, double xCenter, double
    yCenter, int interpolation);
IppStatus ippiRotateCenter_16u_P4R(const Ipp16u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp16u* const
    pDst[4], int dstStep, IppiRect dstROI, double angle, double xCenter,
    double yCenter, int interpolation);
IppStatus ippiRotateCenter_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstROI, double angle, double xCenter,
    double yCenter, int interpolation);
```

## Shear

Performs shearing transform of the source image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiShear_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
```

```
IppStatus ippiShear_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
IppStatus ippiShear_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect dstROI,
    double xShear, double yShear, double xShift, double yShift, int
    interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiShear_8u_P3R(const Ipp8u* const pSrc[3], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int dstStep,
    IppiRect dstROI, double xShear, double yShear, double xShift, double
    yShift, int interpolation);
IppStatus ippiShear_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[3], int
    dstStep, IppiRect dstROI, double xShear, double yShear, double
    xShift, double yShift, int interpolation);
IppStatus ippiShear_8u_P4R(const Ipp8u* const pSrc[4], IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int dstStep,
    IppiRect dstROI, double xShear, double yShear, double xShift, double
    yShift, int interpolation);
IppStatus ippiShear_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[4], int
    dstStep, IppiRect dstROI, double xShear, double yShear, double
    xShift, double yShift, int interpolation);
```

## GetShearQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the shearing transform.

```
IppStatus ippiGetShearQuad(IppiRect srcRoi, double quad[4][2], double
    xShear, double yShear, double xShift, double yShift);
```

## GetShearBound

Computes the bounding rectangle for the source ROI transformed by the `ippiShear` function.

```
IppStatus ippiGetShearBound(IppiRect srcRoi, double bound[2][2], double
    xShear, double yShear, double xShift, double yShift);
```

## WarpAffine

Performs general affine transform of the source image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffine_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_16u_C1R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_16u_C3R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_16u_C4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize, int
    srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect dstROI,
    const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_16u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffine_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffine_8u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
```

```

IppStatus ippiWarpAffine_16u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffine_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffine_8u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffine_16u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffine_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);

```

## WarpAffineBack

Performs an inverse affine transform of an image.

### Case 1: Operation on pixel-order data

```

IppStatus ippiWarpAffineBack_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_16u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_16u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);

```



```
IppStatus ippiWarpAffineBack_16u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_16u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][3], int interpolation);
IppStatus ippiWarpAffineBack_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][3], int interpolation);
```

**Case 2: Operation on planar-order data**

```
IppStatus ippiWarpAffineBack_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffineBack_16u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffineBack_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffineBack_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffineBack_16u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
IppStatus ippiWarpAffineBack_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstROI, const double coeffs[2][3], int
    interpolation);
```

## WarpAffineQuad

Performs image affine warping that transforms the given source quadrangle to the specified destination quadrangle.

### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffineQuad_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_16u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);

IppStatus ippiWarpAffineQuad_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_16u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);

IppStatus ippiWarpAffineQuad_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_16u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);

IppStatus ippiWarpAffineQuad_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);

IppStatus ippiWarpAffineQuad_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, const double srcQuad[4][2], Ipp8u*
    pDst, int dstStep, IppiRect dstROI, const double dstQuad[4][2], int
    interpolation);
```

```
IppStatus ippiWarpAffineQuad_16u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffineQuad_8u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* const pDst[3], int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_16u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* const pDst[3], int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[3], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_8u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* const pDst[4], int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_16u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* const pDst[4], int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpAffineQuad_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[4], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
```

## GetAffineQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the affine transform.

```
IppStatus ippiGetAffineQuad(IppiRect srcRoi, double quad[4][2], const
    double coeffs[2][3]);
```

## GetAffineBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

```
IppStatus ippiGetAffineBound(IppiRect srcRoi, double bound[2][2], const
    double coeffs[2][3]);
```

## GetAffineTransform

Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

```
IppStatus ippiGetAffineTransform(IppiRect srcRoi, const double
    quad[4][2], double coeffs[2][3]);
```

## WarpPerspective

Performs perspective warping of the source image using the given transform coefficients.

### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspective_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_8u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspective_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspective_8u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
IppStatus ippiWarpPerspective_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
```

```
IppStatus ippiWarpPerspective_8u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
IppStatus ippiWarpPerspective_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
```

## WarpPerspectiveBack

Performs an inverse perspective warping of the source image.

### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveBack_8u_C1R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_8u_C3R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_8u_C4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_8u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
IppStatus ippiWarpPerspectiveBack_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[3][3], int interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveBack_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
IppStatus ippiWarpPerspectiveBack_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
```

```

IppStatus ippiWarpPerspectiveBack_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);
IppStatus ippiWarpPerspectiveBack_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstROI, const double coeffs[3][3], int
    interpolation);

```

## WarpPerspectiveQuad

Performs perspective warping of the given source quadrangle to the specified destination quadrangle.

### Case 1: Operation on pixel-order data

```

IppStatus ippiWarpPerspectiveQuad_8u_C1R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_8u_C3R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_8u_C4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_8u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpPerspectiveQuad_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);

```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveQuad_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp8u* const pDst[3], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);

IppStatus ippiWarpPerspectiveQuad_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[3], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);

IppStatus ippiWarpPerspectiveQuad_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp8u* const pDst[4], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);

IppStatus ippiWarpPerspectiveQuad_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[4], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
```

### GetPerspectiveQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the perspective transform.

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[3][3]);
```

### GetPerspectiveBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.

```
IppStatus ippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[3][3]);
```

### GetPerspectiveTransform

Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

```
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi, const double
    quad[4][2], double coeffs[3][3]);
```

### WarpBilinear

Performs bilinear warping of the source image using the specified transform coefficients.

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinear_8u_C1R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
```

```

IppStatus ippiWarpBilinear_32f_C1R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_8u_C3R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_32f_C3R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_8u_C4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_32f_C4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_8u_AC4R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinear_32f_AC4R(const Ipp32f* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep, IppiRect
    dstROI, const double coeffs[2][4], int interpolation);

```

### Case 2: Operation on planar-order data

```

IppStatus ippiWarpBilinear_8u_P3R(const Ipp8u* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinear_32f_P3R(const Ipp32f* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[3], int
    dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinear_8u_P4R(const Ipp8u* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinear_32f_P4R(const Ipp32f* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* const pDst[4], int
    dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);

```

## WarpBilinearBack

Performs an inverse bilinear warping of the source image.

### Case 1: Operation on pixel-order data

```

IppStatus ippiWarpBilinearBack_8u_C1R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);

```



```
IppStatus ippiWarpBilinearBack_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_8u_C3R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_8u_C4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_8u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
IppStatus ippiWarpBilinearBack_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, Ipp32f* pDst, int dstStep,
    IppiRect dstROI, const double coeffs[2][4], int interpolation);
```

## **Case 2: Operation on planar-order data**

```
IppStatus ippiWarpBilinearBack_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[3],
    int dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinearBack_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[3], int dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinearBack_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* const pDst[4],
    int dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
IppStatus ippiWarpBilinearBack_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp32f* const
    pDst[4], int dstStep, IppiRect dstROI, const double coeffs[2][4], int
    interpolation);
```

## WarpBilinearQuad

Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearQuad_8u_C1R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_C1R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_8u_C3R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_C3R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_8u_C4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_C4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_8u_AC4R(const Ipp8u* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp8u* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_AC4R(const Ipp32f* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcROI, const double srcQuad[4][2],
    Ipp32f* pDst, int dstStep, IppiRect dstROI, const double
    dstQuad[4][2], int interpolation);
```

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinearQuad_8u_P3R(const Ipp8u* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp8u* const pDst[3], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_P3R(const Ipp32f* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[3], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
```

```
IppStatus ippiWarpBilinearQuad_8u_P4R(const Ipp8u* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp8u* const pDst[4], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
IppStatus ippiWarpBilinearQuad_32f_P4R(const Ipp32f* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcROI, const double
    srcQuad[4][2], Ipp32f* const pDst[4], int dstStep, IppiRect dstROI,
    const double dstQuad[4][2], int interpolation);
```

### GetBilinearQuad

Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the bilinear transform.

```
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2], const
    double coeffs[2][4]);
```

### GetBilinearBound

Computes the bounding rectangle for the source ROI transformed by the `ippiWarpBilinear` function.

```
IppStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[2][4]);
```

### GetBilinearTransform

Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

```
IppStatus ippiGetBilinearTransform(IppiRect srcRoi, const double
    quad[4][2], double coeffs[2][4]);
```

## Wavelet Transforms

### WTFwdInitAlloc

Allocates memory and initializes the forward wavelet transform context structure.

```
IppStatus ippiWTFwdInitAlloc_32f_C1R (IppiWTFwdSpec_32f_C1R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f*
    pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTFwdInitAlloc_32f_C3R (IppiWTFwdSpec_32f_C3R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f*
    pTapsHigh, int lenHigh, int anchorHigh);
```

## WTFwdFree

Deallocates memory allocated for a forward wavelet transform context structure.

```
IppStatus ippiWTFwdFree_32f_C1R (IppiWTFwdSpec_32f_C1R* pSpec);  
IppStatus ippiWTFwdFree_32f_C3R (IppiWTFwdSpec_32f_C3R* pSpec);
```

## WTFwdGetBufSize

Determines the size of external work buffer for a forward wavelet transform.

```
IppStatus ippiWTFwdGetBufSize_C1R(const IppiWTFwdSpec_32f_C1R* pSpec,  
    int* pSize);  
IppStatus ippiWTFwdGetBufSize_C3R(const IppiWTFwdSpec_32f_C3R* pSpec,  
    int* pSize);
```

## WTFwd

Performs one-level wavelet decomposition of an image.

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,  
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int  
    detailXYStep, IppiSize dstRoiSize, IppiWTFwdSpec_32f_C1R* pSpec,  
    Ipp8u* pBuffer);  
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f*  
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,  
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int  
    detailXYStep, IppiSize dstRoiSize, IppiWTFwdSpec_32f_C3R* pSpec,  
    Ipp8u* pBuffer);
```

## WTInvInitAlloc

Allocates memory and initializes the inverse wavelet transform context structure.

```
IppStatus ippiWTInvInitAlloc_32f_C1R (IppiWTInvSpec_32f_C1R** pSpec,  
    const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f*  
    pTapsHigh, int lenHigh, int anchorHigh);  
IppStatus ippiWTInvInitAlloc_32f_C3R (IppiWTInvSpec_32f_C3R** pSpec,  
    const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f*  
    pTapsHigh, int lenHigh, int anchorHigh);
```

## WTInvFree

Deallocates memory allocated for an inverse wavelet transform context structure.

```
IppStatus ippiWTInvFree_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec);  
IppStatus ippiWTInvFree_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec);
```

## WTInvGetBufSize

Determines the size of external work buffer for an inverse wavelet transform.

```
IppStatus ippiWTInvGetBufSize_C1R( const IppiWTInvSpec_32f_C1R* pSpec,
    int* pSize );

IppStatus ippiWTInvGetBufSize_C3R( const IppiWTInvSpec_32f_C3R* pSpec,
    int* pSize );
```

## WTInv

Performs one-level wavelet reconstruction of an image.

```
IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep,
    const Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f*
    pDetailYsrc, int detailYStep, const Ipp32f* pDetailXYsrc, int
    detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiWTInvSpec_32f_C1R* pSpec, Ipp8u* pBuffer);

IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep,
    const Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f*
    pDetailYsrc, int detailYStep, const Ipp32f* pDetailXYsrc, int
    detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiWTInvSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

# Computer Vision

## Feature Detection Functions

### CannyGetSize

Calculates size of temporary buffer for function `ippiCanny`.

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufSize);
```

### Canny

Implements Canny algorithm for edge detection.

```
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s*  
    pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize  
    roiSize, Ipp32f lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);  
  
IppStatus ippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f*  
    pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize  
    roiSize, Ipp32f lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);
```

### EigenValsVecsGetBufferSize

Calculates size of temporary buffer for the function `ippiEigenValsVecs`.

```
IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize, int  
    apertureSize, int avgWindow, int* pBufferSize);  
  
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int  
    apertureSize, int avgWindow, int* pBufferSize);
```

### EigenValsVecs

Calculates eigen values and eigen vectors of image blocks for corner detection.

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType  
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);  
  
IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType  
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

### MinEigenValGetBufferSize

Calculates size of temporary buffer for the function `ippiMinEigenVal`.

```
IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int  
    apertureSize, int avgWindow, int* pBufferSize);
```

```
IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
```

## MinEigenVal

Calculates the minimal eigen value of image blocks for corner detection.

```
IppStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize,
    IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u*
    pBuffer);

IppStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize,
    IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u*
    pBuffer);
```

## Distance Transform Functions

### DistanceTransform

Calculates distance to closest zero pixel for all non-zero pixels of source image.

#### Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);

IppStatus ippiDistanceTransform_5x5_8u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);

IppStatus ippiDistanceTransform_3x3_8u16u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);

IppStatus ippiDistanceTransform_5x5_8u16u_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);

IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);

IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f*
    pMetrics);
```

#### Case 2: In-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

### GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

```
IppStatus ippiGetDistanceTransformMask_32s(int kerSize, IppiNorm norm,
    Ipp32s* pMetrics);
IppStatus ippiGetDistanceTransformMask_32f(int kerSize, IppiNorm norm,
    Ipp32f* pMetrics);
```

## Image Gradients

### GradientColorToGray

Converts a color gradient image to grayscale.

```
IppStatus ippiGradientColorToGray_8u_C3C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
IppStatus ippiGradientColorToGray_16u_C3C1R(const Ipp16u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
IppStatus ippiGradientColorToGray_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
```

## Flood Fill Functions

### FloodFillGetSize

Calculates size of temporary buffer for function FloodFill.

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

### FloodFillGetSize\_Grad

Calculates size of temporary buffer for function FloodFill\_Grad.

```
ippiStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

### FloodFill

Performs flood filling of connected area on image.

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_4Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```



```
IppStatus ippiFloodFill_8Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_4Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_8Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

**Case 2: Operations on three-channel data**

```
IppStatus ippiFloodFill_4Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_8Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_4Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_8Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

**FloodFill\_Grad**

Performs flood filling of connected area on image.

**Case 1: Operations on one-channel data**

```
IppStatus ippiFloodFill_Grad4Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad8Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad4Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad8Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

**Case 2: Operations on three-channel data**

```
IppStatus ippiFloodFill_Grad4Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad8Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```

IppStatus ippiFloodFill_Grad4Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad8Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

## FloodFill\_Range

Performs flood filling of pixels with values in the specified range in the connected area on an image.

### Case 1: Operations on one-channel data

```

IppStatus ippiFloodFill_Range4Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range8Con_8u_C1IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u newVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range4Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range8Con_32f_C1IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f newVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

### Case 2: Operations on three-channel data

```

IppStatus ippiFloodFill_Range4Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range8Con_8u_C3IR(Ipp8u* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp8u pnewVal, Ipp8u minDelta, Ipp8u
    maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range4Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Range8Con_32f_C3IR(Ipp32f* pImage, int imageStep,
    IppiSize roiSize, IppiPoint seed, Ipp32f pnewVal, Ipp32f minDelta,
    Ipp32f maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

```

## Motion Analysis and Object Tracking

### UpdateMotionHistory

Updates motion history image using motion silhouette at given timestamp.

```

IppStatus ippiUpdateMotionHistory_8u32f_C1IR(const Ipp8u* pSilhouette,
    int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f
    timeStamp, Ipp32f mhiDuration);

```

```
IppStatus ippiUpdateMotionHistory_16u32f_C1R(const Ipp16u* pSilhouette,
    int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f
    timeStamp, Ipp32f mhiDuration);
IppStatus ippiUpdateMotionHistory_8u32f_C1R(const Ipp32f* pSilhouette,
    int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f
    timeStamp, Ipp32f mhiDuration);
```

## OpticalFlowPyrLKInitAlloc

Allocates memory and initializes a structure for optical flow calculation.

```
IppStatus ippiOpticalFlowPyrLKInitAlloc_8u_C1R(IppiOptFlowPyrLK_8u_C1R**
    ppState, IppiSize roiSize, int winSize, IppHintAlgorithm hint);
IppStatus
    ippiOpticalFlowPyrLKInitAlloc_16u_C1R(IppiOptFlowPyrLK_16u_C1R**
    ppState, IppiSize roiSize, int winSize, IppHintAlgorithm hint);
IppStatus ippiOpticalFlowPyrLKInitAlloc_32f_C1R
    (IppiOptFlowPyrLK_32f_C1R** ppState, IppiSize roiSize, int winSize,
    IppHintAlgorithm hint);
```

## OpticalFlowPyrLKFree

Frees memory allocated for the optical flow structure.

```
IppStatus ippiOpticalFlowPyrLKFree_8u_C1R(IppiOptFlowPyrLK_8u_C1R*
    pState);
IppStatus ippiOpticalFlowPyrLKFree_16u_C1R(IppiOptFlowPyrLK_16u_C1R*
    pState);
IppStatus ippiOpticalFlowPyrLKFree_32f_C1R(IppiOptFlowPyrLK_32f_C1R*
    pState);
```

## OpticalFlowPyrLK

Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

```
IppStatus ippiOpticalFlowPyrLK_8u_C1R(IppiPyramid* pPyr1, IppiPyramid*
    pPyr2, const IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s*
    pStatus, Ipp32f* pError, int numFeat, int winSize, int maxLev, int
    maxIter, Ipp32f threshold, IppiOptFlowPyrLK_8u_C1R* pState);
IppStatus ippiOpticalFlowPyrLK_16u_C1R(IppiPyramid* pPyr1, IppiPyramid*
    pPyr2, const IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s*
    pStatus, Ipp32f* pError, int numFeat, int winSize, int maxLev, int
    maxIter, Ipp32f threshold, IppiOptFlowPyrLK_16u_C1R* pState);
```

```
IppStatus ippiOpticalFlowPyrLK_32f_C1R(IppiPyramid* pPyr1, IppiPyramid*
pPyr2, const IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s*
pStatus, Ipp32f* pError, int numFeat, int winSize, int maxLev, int
maxIter, Ipp32f threshold, IppiOptFlowPyrLK_32f_C1R* pState);
```

## Pyramids Functions

### PyrDownGetBufSize

Computes the size of temporary buffer for the function `ippiPyrDown`.

```
IppStatus ippiPyrDownGetBufSize_Gauss5x5(int roiWidth, IppDataType
dataType, int channels, int* pBufSize);
```

### PyrUpGetBufSize

Calculates size of temporary buffer for the function `ippiPyrUp`.

```
IppStatus ippiPyrUpGetBufSize_Gauss5x5(int roiWidth, IppDataType
dataType, int channels, int* pBufSize);
```

### PyrDown

Applies the Gaussian to image and then performs down-sampling.

```
IppStatus ippiPyrDown_Gauss5x5_8u_C1R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrDown_Gauss5x5_8u_C3R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrDown_Gauss5x5_8s_C1R(const Ipp8s* pSrc, int srcStep,
Ipp8s* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrDown_Gauss5x5_8s_C3R(const Ipp8s* pSrc, int srcStep,
Ipp8s* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrDown_Gauss5x5_32f_C1R(const Ipp32f* pSrc, int srcStep,
Ipp32f* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrDown_Gauss5x5_32f_C3R(const Ipp32f* pSrc, int srcStep,
Ipp32f* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
```

### PyrUp

Up-samples image and then applies the Gaussian.

```
IppStatus ippiPyrUp_Gauss5x5_8u_C1R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrUp_Gauss5x5_8u_C3R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrUp_Gauss5x5_8s_C1R(const Ipp8s* pSrc, int srcStep,
Ipp8s* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
```

```
IppStatus ippiPyrUp_Gauss5x5_8s_C3R(const Ipp8s* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrUp_Gauss5x5_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
IppStatus ippiPyrUp_Gauss5x5_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, void* pBuffer);
```

## Universal Pyramids

### PyramidInitAlloc

Allocates memory and initializes a pyramid structure.

```
IppStatus ippiPyramidInitAlloc(IppiPyramid** ppPyr, int level, IppiSize
    roiSize, Ipp32f rate);
```

### PyramidFree

Frees memory allocated for the pyramid structure.

```
IppStatus ippiPyramidFree(IppiPyramid* pPyr);
```

### PyramidLayerDownInitAlloc

Allocates memory and initializes a structure for creating a lower pyramid layer.

#### Case 1: Operation on integer data

```
IppStatus
    ippiPyramidLayerDownInitAlloc_8u_C1R(IppiPyramidDownState_8u_C1R**
        ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int
        kerSize, int mode);
IppStatus
    ippiPyramidLayerDownInitAlloc_8u_C3R(IppiPyramidDownState_8u_C3R**
        ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int
        kerSize, int mode);
IppStatus
    ippiPyramidLayerDownInitAlloc_16u_C1R(IppiPyramidDownState_16u_C1R**
        ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int
        kerSize, int mode);
IppStatus
    ippiPyramidLayerDownInitAlloc_16u_C3R(IppiPyramidDownState_16u_C3R**
        ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int
        kerSize, int mode);
```

**Case 2: Operation on floating point data**

```
IppStatus  
ippiPyramidLayerDownInitAlloc_32f_C1R(IppiPyramidDownState_32f_C1R**  
ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp32f* pKernel, int  
kerSize, int mode);  
  
IppStatus  
ippiPyramidLayerDownInitAlloc_32f_C3R(IppiPyramidDownState_32f_C3R**  
ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp32f* pKernel, int  
kerSize, int mode);
```

**PyramidLayerDownFree**

Frees memory allocated for the lower pyramid layer structure.

```
IppStatus ippiPyramidLayerDownFree_8u_C1R(IppiPyramidDownState_8u_C1R*  
pState);  
  
IppStatus ippiPyramidLayerDownFree_8u_C3R(IppiPyramidDownState_8u_C3R*  
pState);  
  
IppStatus ippiPyramidLayerDownFree_16u_C1R(IppiPyramidDownState_16u_C1R*  
pState);  
  
IppStatus ippiPyramidLayerDownFree_16u_C3R(IppiPyramidDownState_16u_C3R*  
pState);  
  
IppStatus ippiPyramidLayerDownFree_32f_C1R(IppiPyramidDownState_32f_C1R*  
pState);  
  
IppStatus ippiPyramidLayerDownFree_32f_C3R(IppiPyramidDownState_32f_C3R*  
pState);
```

**PyramidLayerUpInitAlloc**

Allocates memory and initializes a structure for creating an upper pyramid layer.

**Case 1: Operation on integer data**

```
IppStatus ippiPyramidLayerUpInitAlloc_8u_C1R(IppiPyramidUpState_8u_C1R**  
ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int  
kerSize, int mode);  
  
IppStatus ippiPyramidLayerUpInitAlloc_8u_C3R(IppiPyramidUpState_8u_C3R**  
ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int  
kerSize, int mode);  
  
IppStatus  
ippiPyramidLayerUpInitAlloc_16u_C1R(IppiPyramidUpState_16u_C1R**  
ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int  
kerSize, int mode);
```

```
IppStatus  
   ippiPyramidLayerUpInitAlloc_16u_C3R(IppiPyramidUpState_16u_C3R**  
    ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int  
    kerSize, int mode);
```

### Case 2: Operation on floating point data

```
IppStatus  
   ippiPyramidLayerUpInitAlloc_32f_C1R(IppiPyramidUpState_32f_C1R**  
    ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp32f* pKernel, int  
    kerSize, int mode);
```

```
IppStatus  
   ippiPyramidLayerUpInitAlloc_32f_C3R(IppiPyramidUpState_32f_C3R**  
    ppState, IppiSize dstRoiSize, Ipp32f rate, Ipp32f* pKernel, int  
    kerSize, int mode);
```

## PyramidLayerUpFree

Frees memory allocated for the upper pyramid layer structure.

```
IppStatus ippiPyramidLayerUpFree_8u_C1R(IppiPyramidUpState_8u_C1R*  
    pState);  
IppStatus ippiPyramidLayerUpFree_8u_C3R(IppiPyramidUpState_8u_C3R*  
    pState);  
IppStatus ippiPyramidLayerUpFree_16u_C1R(IppiPyramidUpState_16u_C1R*  
    pState);  
IppStatus ippiPyramidLayerUpFree_16u_C3R(IppiPyramidUpState_16u_C3R*  
    pState);  
IppStatus ippiPyramidLayerUpFree_32f_C1R(IppiPyramidUpState_32f_C1R*  
    pState);  
IppStatus ippiPyramidLayerUpFree_32f_C3R(IppiPyramidUpState_32f_C3R*  
    pState);
```

## GetPyramidDownROI

Computes the size of the lower pyramid layer.

```
IppStatus ippiGetPyramidDownROI(IppiSize srcRoiSize, IppiSize*  
    pDstRoiSize, Ipp32f rate);
```

## GetPyramidUpROI

Computes the size of the upper pyramid layer.

```
IppStatus ippiGetPyramidUpROI(IppiSize srcRoiSize, IppiSize*  
    pDstRoiSizeMin, IppiSize* pDstRoiSizeMax, Ipp32f rate);
```

## PyramidLayerDown

Creates a lower pyramid layer.

```
IppStatus ippiPyramidLayerDown_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_8u_C1R* pState);

IppStatus ippiPyramidLayerDown_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_8u_C3R* pState);

IppStatus ippiPyramidLayerDown_16u_C1R(const Ipp16u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_16u_C1R* pState);

IppStatus ippiPyramidLayerDown_16u_C3R(const Ipp16u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_16u_C3R* pState);

IppStatus ippiPyramidLayerDown_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_32f_C1R* pState);

IppStatus ippiPyramidLayerDown_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidDownState_32f_C3R* pState);
```

## PyramidLayerUp

Creates an upper pyramid layer.

```
IppStatus ippiPyramidLayerUp_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_8u_C1R* pState);

IppStatus ippiPyramidLayerUp_8u_C3R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_8u_C3R* pState);

IppStatus ippiPyramidLayerUp_16u_C1R(const Ipp16u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_16u_C1R* pState);

IppStatus ippiPyramidLayerUp_16u_C3R(const Ipp16u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_16u_C3R* pState);

IppStatus ippiPyramidLayerUp_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_32f_C1R* pState);
```



```
IppStatus ippiPyramidLayerUp_32f_C3R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_32f_C3R* pState);
```

## Pattern Recognition

### HaarClassifierInitAlloc

Allocates memory and initializes the structure for standard Haar classifiers.

```
IppStatus ippiHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f**
    pState, const IppiRect* pFeature, const Ipp32f* pWeight, const
    Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const
    int* pNum, int length);

IppStatus ippiHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s**
    pState, const IppiRect* pFeature, const Ipp32s* pWeight, const
    Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const
    int* pNum, int length);
```

### TiltedHaarClassifierInitAlloc

Allocates memory and initializes the structure for tilted Haar classifiers.

```
IppStatus ippiTiltedHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f**
    pState, const IppiRect* pFeature, const Ipp32f* pWeight, const
    Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const
    int* pNum, int length);

IppStatus ippiTiltedHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s**
    pState, const IppiRect* pFeature, const Ipp32s* pWeight, const
    Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const
    int* pNum, int length);
```

### HaarClassifierFree

Frees memory allocated for the Haar classifier structure.

```
IppStatus ippiHaarClassifierFree_32f(IppiHaarClassifier_32f* pState);
IppStatus ippiHaarClassifierFree_32s(IppiHaarClassifier_32s* pState);
```

### GetHaarClassifierSize

Returns the size of the Haar classifier.

```
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState,
    IppiSize* pSize);

IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState,
    IppiSize* pSize);
```

## TiltHaarFeatures

Modifies a Haar classifier by tilting specified features.

```
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32f* pState);

IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32s* pState);
```

## ApplyHaarClassifier

Applies a Haar classifier to an image.

```
IppStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int
    srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int
    maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold,
    IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int
    srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int
    maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold,
    IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int
    maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold,
    IppiHaarClassifier_32s* pState);
```

## ApplyMixedHaarClassifier

Applies a mixed Haar classifier to an image.

```
IppStatus ippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int
    srcStep, const Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int
    normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
    pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc,
    int srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm,
    int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
    pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc,
    int srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm,
    int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int*
    pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int
    scaleFactor);
```

## Camera Calibration and 3D Reconstruction

### UndistortGetSize

Computes the size of the external buffer.

```
IppStatus ippiUndistortGetSize(IppiSize roiSize, int* pBufsize);
```

### UndistortRadial

Corrects radial distortions of the single image.

```
IppStatus ippiUndistortRadial_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
IppStatus ippiUndistortRadial_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
IppStatus ippiUndistortRadial_16u_C1R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
IppStatus ippiUndistortRadial_16u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
IppStatus ippiUndistortRadial_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
IppStatus ippiUndistortRadial_32f_C3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
```

### CreateMapCameraUndistort

Creates look-up tables of coordinates of corrected image.

```
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep,
    Ipp32f* pyMap, int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
    Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp32f p1, Ipp32f p2,
    Ipp8u* pBuffer);
```

# Image Compression Functions

## Support Functions

### **ippjGetLibVersion**

Returns information of the library version.

```
const IppLibraryVersion* ippjGetLibVersion(void);
```

## Color Conversion Functions

### **RGBToY\_JPEG**

Converts an RGB image to gray scale.

#### **Case 1: Operation on planar data**

```
IppStatus ippRGBToY_JPEG_8u_P3C1R(const Ipp8u* pSrcRGB[3], int srcStep,  
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

#### **Case 2: Operation on pixel-order data**

```
IppStatus ippiRGBToY_JPEG_8u_C3C1R(const Ipp8u* pSrcRGB, int srcStep,  
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

### **BGRToY\_JPEG**

Converts a BGR image to gray scale.

```
IppStatus ippiBGRToY_JPEG_8u_C3C1R(const Ipp8u* pSrcBGR, int srcStep,  
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

### **RGBToYCbCr\_JPEG**

Converts an RGB image to YCbCr color model.

#### **Case 1: Operation on planar data**

```
IppStatus ippRGBToYCbCr_JPEG_8u_P3R(const Ipp8u* pSrcRGB[3], int  
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

#### **Case 2: Operation on pixel-order data**

```
IppStatus ippiRGBToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcRGB, int  
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

## YCbCrToRGB\_JPEG

Converts an YCbCr image to the RGB color model.

### Case 1: Operation on planar data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstRGB[3], int dstStep, IppiSize roiSize);
```

### Case 2: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstRGB, int dstStep, IppiSize roiSize);
```

## RGB565ToYCbCr\_JPEG, RGB555ToYCbCr\_JPEG

Convert an RGB image to YCbCr color model.

```
IppStatus ippiRGB565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
IppStatus ippiRGB555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

## YCbCrToRGB565\_JPEG, YCbCrToRGB555\_JPEG

Convert an YCbCr image to the RGB color model.

```
IppStatus ippiYCbCrToRGB565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);
```

## BGRToYCbCr\_JPEG

Converts a BGR image to YCbCr color model.

```
IppStatus ippiBGRToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcBGR, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

## YCbCrToBGR\_JPEG

Converts an YCbCr image to the BGR color model.

```
IppStatus ippiYCbCrToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstBGR, int dstStep, IppiSize roiSize);
```

## **BGR565ToYCbCr\_JPEG, BGR555ToYCbCr\_JPEG**

Convert a BGR image to YCbCr color model.

```
IppStatus ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);

IppStatus ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

## **YCbCrToBGR565\_JPEG, YCbCrToBGR555\_JPEG**

Convert an YCbCr image to the BGR color model.

```
IppStatus ippiYCbCrToBGR565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCrToBGR555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
```

## **CMYKToYCKK\_JPEG**

Converts a CMYK image to the YCKK color model.

### **Case 1: Operation on planar data**

```
IppStatus ippiCMYKToYCKK_JPEG_8u_P4R(const Ipp8u* pSrcCMYK[4], int
    srcStep, Ipp8u* pDstYCKK[4], int dstStep, IppiSize roiSize);
```

### **Case 2: Operation on pixel-order data**

```
IppStatus ippiCMYKToYCKK_JPEG_8u_C4P4R(const Ipp8u* pSrcCMYK, int
    srcStep, Ipp8u* pDstYCKK[4], int dstStep, IppiSize roiSize);
```

## **YCKKToCMYK\_JPEG**

Converts an YCKK image to the CMYK color model.

```
IppStatus ippiYCKKToCMYK_JPEG_8u_P4R(const Ipp8u* pSrcYCKK[4], int
    srcStep, Ipp8u* pDstCMYK[4], int dstStep, IppiSize roiSize);

IppStatus ippiYCKKToCMYK_JPEG_8u_P4C4R(const Ipp8u* pSrcYCKK[4], int
    srcStep, Ipp8u* pDstCMYK, int dstStep, IppiSize roiSize);
```

## Combined Color Conversion Functions

### RGBToYCbCr444LS\_MCU

Converts an RGB image to the YCbCr color model and creates 444 MCU.

```
IppStatus ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int
    srcStep, Ipp16s* pDstMCU[3]);
```

### RGBToYCbCr422LS\_MCU

Converts an RGB image to the YCbCr color model and creates 422 MCU.

```
IppStatus ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int
    srcStep, Ipp16s* pDstMCU[3]);
```

### RGBToYCbCr411LS\_MCU

Converts an RGB image to the YCbCr color model and creates 411 MCU.

```
IppStatus ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB, int
    srcStep, Ipp16s* pDstMCU[3]);
```

### BGRToYCbCr444LS\_MCU

Converts a BGR image to the YCbCr color model and creates 444 MCU.

```
IppStatus ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int
    srcStep, Ipp16s* pDstMCU[3]);
```

### BGR565ToYCbCr444LS\_MCU, BGR555ToYCbCr444LS\_MCU

Convert a BGR image to the YCbCr color model and create 444 MCU.

```
IppStatus ippiBGR565ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

```
IppStatus ippiBGR555ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

### BGRToYCbCr422LS\_MCU

Converts an BGR image to the YCbCr color model and creates 422 MCU.

```
IppStatus ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int
    srcStep, Ipp16s* pDstMCU[3]);
```

**BGR565ToYCbCr422LS\_MCU,  
BGR555ToYCbCr422LS\_MCU**

Convert an BGR image to the YCbCr color model and create 422 MCU.

```
IppStatus ippkBGR565ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);

IppStatus ippkBGR555ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

**BGRToYCbCr411LS\_MCU**

Converts an BGR image to the YCbCr color model and creates 411 MCU.

```
IppStatus ippkBGRToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR, int
    srcStep, Ipp16s* pDstMCU[3]);
```

**BGR565ToYCbCr411LS\_MCU,  
BGR555ToYCbCr411LS\_MCU**

Convert an BGR image to the YCbCr color model and create 411 MCU.

```
IppStatus ippkBGR565ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);

IppStatus ippkBGR555ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

**CMYKToYCKK444LS\_MCU**

Converts a CMYK image to the YCKK color model and creates 4444 MCU.

```
IppStatus ippbCMYKToYCKK444LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

**CMYKToYCKK422LS\_MCU**

Converts a CMYK image to the YCKK color model and creates 4224 MCU.

```
IppStatus ippbCMYKToYCKK422LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

**CMYKToYCKK411LS\_MCU**

Converts a CMYK image to the YCKK color model and creates 4114 MCU.

```
IppStatus ippbCMYKToYCKK411LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```



### **YCbCr444ToRGBLS\_MCU**

Creates an YCbCr image from 444 MCU and converts it to the RGB color model.

```
IppStatus ippiYCbCr444ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstRGB, int dstStep);
```

### **YCbCr422ToRGBLS\_MCU**

Creates an YCbCr image from 422 MCU and converts it to the RGB color model.

```
IppStatus ippiYCbCr422ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstRGB, int dstStep);
```

### **YCbCr411ToRGBLS\_MC**

Creates an YCbCr image from 411 MCU and converts it to the RGB color model.

```
IppStatus ippiYCbCr411ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstRGB, int dstStep);
```

### **YCbCr444ToBGRLS\_MCU**

Creates an YCbCr image from 444 MCU and converts it to the BGR color model.

```
IppStatus ippiYCbCr444ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstBGR, int dstStep);
```

### **YCbCr444ToBGR565LS\_MCU, YCbCr444ToBGR555LS\_MCU**

Create an YCbCr image from 444 MCU and convert it to the BGR color model.

```
IppStatus ippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s*  
        pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);  
IppStatus ippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s*  
        pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

### **YCbCr422ToBGRLS\_MCU**

Creates a YCbCr image from 422 MCU and converts it to the BGR color model.

```
IppStatus ippiYCbCr422ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstBGR, int dstStep);
```

### **YCbCr422ToBGR565LS\_MCU, YCbCr422ToBGR555LS\_MCU**

Create an YCbCr image from 422 MCU and convert it to the BGR color model.

```
IppStatus ippiYCbCr422ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s*  
        pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

```
IppStatus ippiYCbCr422ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s*  
    pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

### **YCbCr411ToBGR555LS\_MCU**

Creates an YCbCr image from 411 MCU and converts it to the BGR color model.

```
IppStatus ippiYCbCr411ToBGR555LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
    Ipp8u* pDstBGR, int dstStep);
```

### **YCbCr411ToBGR565LS\_MCU, YCbCr411ToBGR555LS\_MCU**

Create an YCbCr image from 411 MCU and convert it to the BGR color model.

```
IppStatus ippiYCbCr411ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s*  
    pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);  
  
IppStatus ippiYCbCr411ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s*  
    pSrcMCU[3], Ipp16u* pDstBGR, int dstStep);
```

### **YCK444ToCMYKLS\_MCU**

Creates YCK image from 4444 MCU and converts it to the CMYK color model.

```
IppStatus ippiYCK444ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
    Ipp8u* pDstCMYK, int dstStep);
```

### **YCK422ToCMYKLS\_MCU**

Creates an YCK image from 4224 MCU and converts it to the CMYK color model.

```
IppStatus ippiYCK422ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
    Ipp8u* pDstCMYK, int dstStep);
```

### **YCK411ToCMYKLS\_MCU**

Creates an YCK image from 4114 MCU and converts it to the CMYK color model.

```
IppStatus ippiYCK411ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
    Ipp8u* pDstCMYK, int dstStep);
```

## **Quantization Functions**

### **QuantFwdRawTableInit\_JPEG**

Modifies raw quantization tables in accordance with the quality factor.

```
IppStatus ippiQuantFwdRawTableInit_JPEG_8u(Ipp8u* pQuantRawTable, int  
    qualityFactor);
```

### QuantFwdTableInit\_JPEG

Prepares quantization tables suitable for fast encoding.

```
IppStatus ippiQuantFwdTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable,  
      Ipp16u* pQuantFwdTable);
```

### QuantFwd8x8\_JPEG

Performs quantization of an 8x8 block of DCT coefficients.

```
IppStatus ippiQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*  
      pQuantFwdTable);
```

### QuantInvTableInit\_JPEG

Prepares quantization tables suitable for fast decoding.

```
IppStatus ippiQuantInvTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable,  
      Ipp16u* pQuantInvTable);
```

### QuantInv8x8\_JPEG

Performs dequantization of an 8x8 block of DCT coefficients

```
IppStatus ippiQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*  
      pQuantInvTable);
```

## Combined DCT Functions

### DCTQuantFwd8x8\_JPEG

Performs forward DCT and quantization.

```
IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*  
      pDst, const Ipp16u* pQuantFwdTable);  
  
IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*  
      pQuantFwdTable);
```

### DCTQuantFwd8x8LS\_JPEG

Performs level shift, forward DCT and quantization.

```
IppStatus ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R(const Ipp8u* pSrc, int  
      srcStep, Ipp16s* pDst, const Ipp16u* pQuantFwdTable);
```

### DCTQuantInv8x8\_JPEG

Performs dequantization and inverse DCT.

```
IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, const Ipp16u* pQuantInvTable);

IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantInvTable);
```

### DCTQuantInv8x8LS\_JPEG

Performs dequantization, inverse DCT and level shift.

```
IppStatus ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstStep, const Ipp16u* pQuantInvTable);
```

## Level Shift Functions

### Sub128\_JPEG

Converts data from the unsigned 8u range to the signed 16s range.

```
IppStatus ippiSub128_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
```

### Add128\_JPEG

Restores samples to the unsigned representation.

```
IppStatus ippiAdd128_JPEG_16s8u_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Sampling Functions

### SampleDownH2V1\_JPEG

Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.

```
IppStatus ippiSampleDownH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### SampleDownH2V2\_JPEG

Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.

```
IppStatus ippiSampleDownH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### SampleDownRowH2V1\_Box\_JPEG

Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.

```
IppStatus ippiSampleDownRowH2V1_Box_JPEG_8u_C1(const Ipp8u* pSrc, int
    srcWidth, Ipp8u* pDst);
```

### SampleDownRowH2V2\_Box\_JPEG

Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.

```
IppStatus ippiSampleDownRowH2V2_Box_JPEG_8u_C1(const Ipp8u* pSrc1, const
    Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

### SampleUpH2V1\_JPEG

Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.

```
IppStatus ippiSampleUpH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### SampleUpH2V2\_JPEG

Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.

```
IppStatus ippiSampleUpH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### SampleUpRowH2V1\_Triangle\_JPEG

Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.

```
IppStatus ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1(const Ipp8u* pSrc, int
    srcWidth, Ipp8u* pDst);
```

### SampleUpRowH2V2\_Triangle\_JPEG

Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.

```
IppStatus ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1(const Ipp8u* pSrc1,
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

### SampleDown444LS\_MCU

Creates 444 MCU with level shift from pixel-order data.

```
IppStatus ippiSampleDown444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]);
```

### **SampleDown422LS\_MCU**

Creates 422 MCU with level shift from pixel-order data.

```
IppStatus ippISampleDown422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,  
        int srcStep, Ipp16s* pDstMCU[3]);
```

### **SampleDown411LS\_MCU**

Creates 411 MCU with level shift from pixel-order data.

```
IppStatus ippISampleDown411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,  
        int srcStep, Ipp16s* pDstMCU[3]);
```

### **SampleUp444LS\_MCU**

Creates pixel-order image from 444 MCU and performs level shift.

```
IppStatus ippISampleUp444LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDst, int dstStep);
```

### **SampleUp422LS\_MCU**

Creates pixel-order image from 422 MCU and performs level shift.

```
IppStatus ippISampleUp422LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDst, int dstStep);
```

### **SampleUp411LS\_MCU**

Creates pixel-order image from 411 MCU and performs level shift.

```
IppStatus ippISampleUp411LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDst, int dstStep);
```

## **Planar-to-Pixel and Pixel-to-Planar Conversion Functions**

### **Split422LS\_MCU**

Creates 422 MCU from 422 interleaved data with level shift.

```
IppStatus ippISplit422LS_MCU_8u16s_C2P3R(const Ipp8u* pSrc, int SrcStep,  
        Ipp16s* pDstMCU[3]);
```

### **Join422LS\_MCU**

Creates 422 interleaved data from 422 MCU with level shift.

```
IppStatus ippIJoin422LS_MCU_16s8u_P3C2R(const Ipp16s* pSrcMCU[3], Ipp8u*  
        pDst, int dstStep);
```

## Huffman Codec Functions

### EncodeHuffmanRawTableInit\_JPEG

Creates raw Huffman tables using Huffman symbols statistics.

```
IppStatus ippiEncodeHuffmanRawTableInit_JPEG_8u(const int  
    pStatistics[256], Ipp8u* pListBits, Ipp8u* pListVals);
```

### EncodeHuffmanSpecGetBufSize\_JPEG

Returns the length of the `IppiEncodeHuffmanSpec` structure.

```
IppStatus ippiEncodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

### EncodeHuffmanSpecInit\_JPEG

Creates Huffman table in a format that is suitable for an encoder.

```
IppStatus ippiEncodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,  
    const Ipp8u* pListVals, IppiEncodeHuffmanSpec* pEncHuffSpec);
```

### EncodeHuffmanSpecInitAlloc\_JPEG

Allocates memory and creates Huffman table in a format that is suitable for an encoder.

```
IppStatus ippiEncodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,  
    const Ipp8u* pListVals, IppiEncodeHuffmanSpec** ppEncHuffSpec);
```

### EncodeHuffmanSpecFree\_JPEG

Frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG_8u` function.

```
IppStatus ippiEncodeHuffmanSpecFree_JPEG_8u  
    (IppiEncodeHuffmanSpec* pEncHuffSpec);
```

### EncodeHuffmanStateGetBufSize\_JPEG

Returns the length of the `IppiEncodeHuffmanState` structure.

```
IppStatus ippiEncodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

### EncodeHuffmanStateInit\_JPEG

Initializes the `IppiEncodeHuffmanState` structure.

```
IppStatus ippiEncodeHuffmanStateInit_JPEG_8u( IppiEncodeHuffmanState*  
    pEncHuffState);
```

### EncodeHuffmanStateInitAlloc\_JPEG

Allocates memory and initializes the `IppiEncodeHuffmanState` structure.

```
IppStatus  
    ippiEncodeHuffmanStateInitAlloc_JPEG_8u(IppiEncodeHuffmanState**  
        pEncHuffState);
```

### EncodeHuffmanStateFree\_JPEG

Frees memory allocated by the `ippiEncodeHuffmanStateInitAlloc_JPEG` function.

```
IppStatus ippiEncodeHuffmanStateFree_JPEG_8u(IppiEncodeHuffmanState*  
    pEncHuffState);
```

### EncodeHuffman8x8\_JPEG

Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.

```
IppStatus ippiEncodeHuffman8x8_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u*  
    pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, const  
    IppiEncodeHuffmanSpec* pDcTable, const IppiEncodeHuffmanSpec*  
    pAcTable, IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### EncodeHuffman8x8\_Direct\_JPEG

Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.

```
IppStatus ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1(const Ipp16s* pSrc,  
    Ipp8u* pDst, int* pDstBitsLen, Ipp16s* pLastDC, const  
    IppiEncodeHuffmanSpec* pDcTable, const IppiEncodeHuffmanSpec*  
    pAcTable);
```

### GetHuffmanStatistics8x8\_JPEG

Computes Huffman symbols statistics for the baseline encoding.

```
IppStatus ippiGetHuffmanStatistics8x8_JPEG_16s_C1(const Ipp16s* pSrc,  
    int pDcStatistics[256], int pAcStatistics[256], Ipp16s* pLastDC);
```

### GetHuffmanStatistics8x8\_DCFfirst\_JPEG

Computes Huffman symbols statistics for the progressive encoding (DC coefficients).

```
IppStatus ippiGetHuffmanStatistics8x8_DCFfirst_JPEG_16s_C1(const Ipp16s*  
    pSrc, int pDcStatistics[256], Ipp16s* pLastDC, int A1);
```

### GetHuffmanStatistics8x8\_ACFfirst\_JPEG

Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).

```
IppStatus ippiGetHuffmanStatistics8x8_ACFfirst_JPEG_16s_C1(const Ipp16s*  
    pSrc, int pAcStatistics[256], int Ss, int Se, int A1,  
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```



### **GetHuffmanStatistics8x8\_ACRefine\_JPEG**

Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).

```
IppStatus ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1(const Ipp16s*  
    pSrc, int pAcStatistics[256], int Ss, int Se, int Al,  
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### **EncodeHuffman8x8\_DCFirst\_JPEG**

Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).

```
IppStatus ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1(const Ipp16s* pSrc,  
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, int  
    Al, IppiEncodeHuffmanSpec* pDcTable, IppiEncodeHuffmanState*  
    pEncHuffState, int bFlushState);
```

### **EncodeHuffman8x8\_DCRefine\_JPEG**

Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).

```
IppStatus ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1(const Ipp16s*  
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Al,  
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### **EncodeHuffman8x8\_ACFirst\_JPEG**

Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).

```
IppStatus ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1(const Ipp16s* pSrc,  
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int  
    Al, IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState*  
    pEncHuffState, int bFlushState);
```

### **EncodeHuffman8x8\_ACRefine\_JPEG**

Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients.

```
IppStatus ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1(const Ipp16s*  
    pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se,  
    int Al, IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState*  
    pEncHuffState, int bFlushState);
```

### DecodeHuffmanSpecGetBufSize\_JPEG

Returns the length of the `IppiDecodeHuffmanSpec` structure.

```
IppStatus ippiDecodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

### DecodeHuffmanSpecInit\_JPEG

Creates Huffman table in a format that is suitable for a decoder

```
IppiDecodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits, const Ipp8u*  
    pListVals, IppiDecodeHuffmanSpec* pDecHuffSpec);
```

### DecodeHuffmanSpecInitAlloc\_JPEG

Allocates memory and creates Huffman table in a format that is suitable for a decoder.

```
IppStatus ippiDecodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,  
    const Ipp8u* pListVals, IppiDecodeHuffmanSpec** ppDecHuffSpec);
```

### DecodeHuffmanSpecFree\_JPEG

Frees memory allocated by `ippiDecodeHuffmanSpecInitAlloc_JPEG` function.

```
IppStatus ippiDecodeHuffmanSpecFree_JPEG_8u(IppiDecodeHuffmanSpec*  
    pDecHuffSpec);
```

### DecodeHuffmanStateGetBufSize\_JPEG

Returns the length of `IppiDecodeHuffmanState` structure.

```
IppStatus ippiDecodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

### DecodeHuffmanStateInit\_JPEG

Initializes `IppiDecodeHuffmanState` structure.

```
IppStatus ippiDecodeHuffmanStateInit_JPEG_8u(IppiDecodeHuffmanState*  
    pDecHuffState);
```

### DecodeHuffmanStateInitAlloc\_JPEG

Allocates memory and initializes `IppiDecodeHuffmanState` structure.

```
IppStatus  
    ippiDecodeHuffmanStateInitAlloc_JPEG_8u(IppiDecodeHuffmanState**  
        ppDecHuffState);
```

## DecodeHuffmanStateFree\_JPEG

Frees memory allocated by `ippiDecodeHuffmanStateInitAlloc_JPEG` function.

```
IppStatus ippiDecodeHuffmanStateFree_JPEG_8u(IppiDecodeHuffmanState*
    pDecHuffState);
```

## DecodeHuffman8x8\_JPEG

Performs Huffman baseline decoding of 8x8 block of the quantized DCT Coefficients.

```
IppStatus ippiDecodeHuffman8x8_JPEG_1u16s_C1(const Ipp8u* pSrc, int
    srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC, int*
    pMarker, const IppiDecodeHuffmanSpec* pDcTable, const
    IppiDecodeHuffmanSpec* pAcTable, IppiDecodeHuffmanState*
    pDecHuffState);
```

## DecodeHuffman8x8\_Direct\_JPEG

Directly performs Huffman baseline decoding of an 8x8 block of the quantized DCT coefficients.

```
IppStatus ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int* pSrcBitsLen, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker,
    Ipp32u* pPrefetchedBits, int* pNumValidPrefetchedBits, const
    IppiDecodeHuffmanSpec* pDcTable, const IppiDecodeHuffmanSpec*
    pAcTable);
```

## DecodeHuffman8x8\_DCFirst\_JPEG

Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).

```
IppStatus ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC,
    int* pMarker, int A1, IppiDecodeHuffmanSpec* pDcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

## DecodeHuffman8x8\_DCRefine\_JPEG

Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients.

```
IppStatus ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int
    A1, IppiDecodeHuffmanState* pDecHuffState);
```

### **DecodeHuffman8x8\_ACFirst\_JPEG**

Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).

```
IppStatus ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int
    Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

### **DecodeHuffman8x8\_ACRefine\_JPEG**

Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).

```
IppStatus ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int
    Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

## **Functions for Lossless JPEG Coding**

### **DiffPredFirstRow\_JPEG**

Computes the differences between input sample and predictor for the first line.

```
IppStatus ippiDiffPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, int width, int P, int Pt);
```

### **DiffPredRow\_JPEG**

Computes the differences between input sample and predictor for all lines but the first.

```
IppStatus ippiDiffPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const Ipp16s*
    pPrevRow, Ipp16s* pDst, int width, int predictor);
```

### **ReconstructPredFirstRow\_JPEG**

Reconstructs samples from the decoded differences between input samples and predictor for the first line.

```
IppStatus ippiReconstructPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc,
    Ipp16s* pDst, int width, int P, int Pt);
```

## ReconstructPredRow\_JPEG

Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.

```
IppStatus ippiReconstructPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const
      Ipp16s* pPrevRow, Ipp16s* pDst, int width, int predictor);
```

## GetHuffmanStatisticsOne\_JPEG

Computes Huffman symbol statistics.

```
IppStatus ippiGetHuffmanStatisticsOne_JPEG_16s_C1(const Ipp16s* pSrc,
      int pHuffStatistics[256]);
```

## EncodeHuffmanOne\_JPEG

Performs Huffman encoding of one difference.

```
IppStatus ippiEncodeHuffmanOne_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u*
      pDst, int nDstLenBytes, int* pDstCurrPos, const
      IppiEncodeHuffmanSpec* pHuffTable, IppiEncodeHuffmanState*
      pEncHuffState, int bFlushState);
```

## DecodeHuffmanOne\_JPEG

Decodes one Huffman coded difference.

```
IppStatus ippiDecodeHuffmanOne_JPEG_1u16s_C1(const Ipp8u* pSrc, int
      nSrcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, const
      IppiDecodeHuffmanSpec* pHuffTable, IppiDecodeHuffmanState*
      pDecHuffState)
```

# Wavelet Transform Functions

## WTFwdRow\_B53\_JPEG2K

Performs a forward wavelet transform with reversible filter of image rows.

```
IppStatus ippiWTFwdRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int
      srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh, int
      dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);

IppStatus ippiWTFwdRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int
      srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh, int
      dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### WTInvRow\_B53\_JPEG2K

Performs an inverse wavelet transform with reversible filter of image rows.

```
IppStatus ippiWTInvRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow, int
    srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);

IppStatus ippiWTInvRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow, int
    srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

### WTFwdCol\_B53\_JPEG2K

Performs a forward wavelet transform with reversible filter of image columns.

```
IppStatus ippiWTFwdCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);

IppStatus ippiWTFwdCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int
    srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### WTFwdColLift\_B53\_JPEG2K

Performs a single step of forward wavelet transform with reversible filter of image columns.

```
IppStatus ippiWTFwdColStepLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrc0,
    const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDstLow0, const
    Ipp16s* pSrcHigh0, Ipp16s* pDstHigh1, int width);

IppStatus ippiWTFwdColStepLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrc0,
    const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDstLow0, const
    Ipp32s* pSrcHigh0, Ipp32s* pDstHigh1, int width);
```

### WTInvCol\_B53\_JPEG2K

Performs an inverse wavelet transform with reversible filter of image columns.

```
IppStatus ippiWTInvCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow, int
    srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);

IppStatus ippiWTInvCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow, int
    srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

### WTInvColLift\_B53\_JPEG2K

Performs a single step of inverse wavelet transform with reversible filter of image columns.

```
IppStatus ippiWTInvColStepLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrcLow0,
    const Ipp16s* pSrcHigh0, const Ipp16s* pSrcHigh1, const Ipp16s*
    pSrc0, Ipp16s* pDst1, Ipp16s* pDst2, int width);

IppStatus ippiWTInvColStepLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrcLow0,
    const Ipp32s* pSrcHigh0, const Ipp32s* pSrcHigh1, const Ipp32s*
    pSrc0, Ipp32s* pDst1, Ipp32s* pDst2, int width);
```

### WTFwdRow\_D97\_JPEG2K

Performs a forward wavelet transform with irreversible filter of image rows.

```
IppStatus ippiWTFwdRow_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc, int
    srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);

IppStatus ippiWTFwdRow_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc, int
    srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);

IppStatus ippiWTFwdRow_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDstLow, int dstLowStep, Ipp32f* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### WTInvRow\_D97\_JPEG2K

Performs an inverse wavelet transform with irreversible filter of image rows.

```
IppStatus ippiWTInvRow_D97_JPEG2K_16s_C1R(const Ipp16s* pSrcLow, int
    srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);

IppStatus ippiWTInvRow_D97_JPEG2K_32s_C1R(const Ipp32s* pSrcLow, int
    srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);

IppStatus ippiWTInvRow_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow, int
    srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, IppiWTFilterFirst phase);
```

### WTFwdCol\_D97\_JPEG2K

Performs a forward wavelet transform with irreversible filter of image columns.

```
IppStatus ippiWTFwdCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDstLow, int dstLowStep, Ipp32f* pDstHigh, int
    dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### WTFwdColLift\_D97\_JPEG2K

Performs a single step of forward wavelet transform with irreversible filter of image columns.

```
IppStatus ippiWTFwdColLift_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc0, const
    Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pSrcDstLow0, Ipp16s*
    pDstLow1, Ipp16s* pSrcDstHigh0, Ipp16s* pSrcDstHigh1, Ipp16s*
    pDstHigh2, int width);

IppStatus ippiWTFwdColLift_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc0, const
    Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pSrcDstLow0, Ipp32s*
    pDstLow1, Ipp32s* pSrcDstHigh0, Ipp32s* pSrcDstHigh1, Ipp32s*
    pDstHigh2, int width);

IppStatus ippiWTFwdColLift_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc0, const
    Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pSrcDstLow0, Ipp32f*
    pDstLow1, Ipp32f* pSrcDstHigh0, Ipp32f* pSrcDstHigh1, Ipp32f*
    pDstHigh2, int width);
```

### WTInvCol\_D97\_JPEG2K

Performs an inverse wavelet transform with irreversible filter of image columns.

```
IppStatus ippiWTInvCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow, int
    srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, IppiWTFilterFirst phase);
```

### WTInvColLift\_D97\_JPEG2K

Performs a single step of inverse wavelet transform with irreversible filter of image columns.

```
IppStatus ippiWTInvColLift_D97_JPEG2K_16s_C1R(const Ipp16s* pSrcLow0,
    const Ipp16s* pSrcHigh0, const Ipp16s* pSrcHigh1, const Ipp16s*
    pSrc0, Ipp16s* pSrcDst1, Ipp16s* pSrcDst2, Ipp16s* pDst3, Ipp16s*
    pDst4, int width);

IppStatus ippiWTInvColLift_D97_JPEG2K_32s_C1R(const Ipp32s* pSrcLow0,
    const Ipp32s* pSrcHigh0, const Ipp32s* pSrcHigh1, const Ipp32s*
    pSrc0, Ipp32s* pSrcDst1, Ipp32s* pSrcDst2, Ipp32s* pDst3, Ipp32s*
    pDst4, int width);

IppStatus ippiWTInvColLift_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow0,
    const Ipp32f* pSrcHigh0, const Ipp32f* pSrcHigh1, const Ipp32f*
    pSrc0, Ipp32f* pSrcDst1, Ipp32f* pSrcDst2, Ipp32f* pDst3, Ipp32f*
    pDst4, int width);
```



## WTGetBufSize\_B53\_JPEG2K

Computes the size of the buffer for wavelet transform with reversible filter.

```
IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1IR(const IppiRect*
        pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1IR(const IppiRect*
        pTileRect, int* pSize);
```

## WTFwd\_B53\_JPEG2K

Performs a tile-oriented forward wavelet transform with reversible filter.

```
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
        const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u*
        pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
        const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp8u*
        pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## WTInv\_B53\_JPEG2K

Performs a tile-oriented inverse wavelet transform with reversible filter.

```
IppStatus ippiWTInv_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc[4], int
        srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
        Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc[4], int
        srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
        Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## WTGetBufSize\_D97\_JPEG2K

Computes the size of the buffer wavelet transform with irreversible filter

```
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1IR(const IppiRect*
        pTileRect, int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1IR(const IppiRect*
        pTileRect, int* pSize);
```

## WTFwd\_D97\_JPEG2K

Performs a tile-oriented forward wavelet transform with irreversible filter.

```
IppStatus ippiWTFwd_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
        const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u*
        pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
        const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp32s*
        pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## WTInv\_D97\_JPEG2K

Performs a tile-oriented inverse wavelet transform with irreversible filter.

```
IppStatus ippiWTInv_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc[4], int
        srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
        Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc[4], int
        srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
        Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
        srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## JPEG2000 Entropy Coding and Decoding Functions

### EncodeInitAlloc\_JPEG2K

Allocates memory and initializes an entropy encoder state structure.

```
IppStatus ippiEncodeInitAlloc_JPEG2K(IppiEncodeState_JPEG2K** pState,  
    IppiSize codeBlockMaxSize);
```

### EncodeFree\_JPEG2K

Frees memory allocated for an entropy encoding state structure.

```
IppStatus ippiEncodeFree_JPEG2K(IppiEncodeState_JPEG2K* pState);
```

### EncodeLoadCodeBlock\_JPEG2K

Loads a code block and prepares data for entropy encoding.

```
IppStatus ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R(const Ipp32s* pSrc, int  
    srcStep, IppiSize codeBlockSize, IppiEncodeState_JPEG2K* pState,  
    IppiWTSubband subband, int magnBits, IppiMQTermination mqTermType,  
    IppiMQRateAppr mqRateAppr, int codeStyleFlags, int* pSfBits, int*  
    pNOFPasses, int* pNOFTermPasses);
```

### EncodeStoreBits\_JPEG2K

Produces output codestream.

```
IppStatus ippiEncodeStoreBits_JPEG2K_1u(Ipp8u* pDst, int* pDstLen,  
    IppiEncodeState_JPEG2K* pState, int* pIsNotFinish);
```

### EncodeGetTermPassLen\_JPEG2K

Returns the length of the specified terminated coding pass.

```
IppStatus ippiEncodeGetTermPassLen_JPEG2K(IppiEncodeState_JPEG2K*  
    pState, int passNumber, int* pPassLen);
```

### EncodeGetRate\_JPEG2K

Returns estimated bit rate for the specified coding pass.

```
IppStatus ippiEncodeGetRate_JPEG2K(IppiEncodeState_JPEG2K* pState, int  
    passNumber, int* pRate);
```

### EncodeGetDist\_JPEG2K

Returns estimated distortion of the image for the specified coding pass.

```
IppStatus ippiEncodeGetDist_JPEG2K(IppiEncodeState_JPEG2K* pState, int  
    passNumber, Ipp64f* pDist);
```

### **DecodeGetBufSize\_JPEG2K**

Computes the size of the work buffer for decoding routine.

```
IppStatus ippidecodeGetBufSize_JPEG2K(IppiSize codeBlockSize, int*
    pSize);
```

### **DecodeCodeBlock\_JPEG2K**

Decodes the compressed code block data.

```
IppStatus ippidecodeCodeBlock_JPEG2K_1u32s_C1R(const Ipp8u* pSrc,
    Ipp32s* pDst, int dstStep, IppiSize codeBlockSize, IppiWTSubband
    subband, int sfBits, int nOfPasses, const int* pTermPassLen, int
    nOfTermPasses, int codeStyleFlags, int* pErrorBitPlane, Ipp8u*
    pBuffer);
```

### **DecodeCBProgrGetStateSize\_JPEG2K**

Computes the size of the external buffer for the decoder state structure.

```
IppStatus ippidecodeCBProgrGetStateSize_JPEG2K(IppiSize
    codeBlockSize, int* pStateSize);
```

### **DecodeCBProgrInit\_JPEG2K**

Initializes the decoder state structure in the external buffer.

```
IppStatus ippidecodeCBProgrInit_JPEG2K(IppiDecodeCBProgrState_JPEG2K*
    pState);
```

### **DecodeCBProgrInitAlloc\_JPEG2K**

Allocates memory and initializes the decoder state structure in the external buffer.

```
IppStatus
    ippidecodeCBProgrInitAlloc_JPEG2K(IppiDecodeCBProgrState_JPEG2K**
    ppState, IppiSize codeBlockSize);
```

### **DecodeCBProgrFree\_JPEG2K**

Frees memory allocated for the decoder state structure.

```
IppStatus ippidecodeCBProgrFree_JPEG2K(IppiDecodeCBProgrState_JPEG2K*
    pState);
```

### **DecodeCBProgrAttach\_JPEG2K**

Attaches the buffer with data for progressive decoding.

```
IppStatus ippidecodeCBProgrAttach_JPEG2K_32s_C1R(Ipp32s* pDst, int
    dstStep, IppiSize codeBlockSize, IppiDecodeCBProgrState_JPEG2K*
    pState, IppiWTSubband subband, int sfBits, int codeStyleFlags);
```

### DecodeCBProgrSetPassCounter\_JPEG2K

Sets the value of internal coding pass counter.

```
IppStatus ippiDecodeCBProgrSetPassCounter_JPEG2K(int nOfPasses,
        IppiDecodeCBProgrState_JPEG2K* pState);
```

### DecodeCBProgrGetPassCounter\_JPEG2K

Returns the value of the internal coding pass counter.

```
IppStatus
    ippiDecodeCBProgrGetPassCounter_JPEG2K(IppiDecodeCBProgrState_JPEG2K
        * pState, int* pNOfResidualPasses);
```

### DecodeCBProgrGetCurBitPlane\_JPEG2K

Returns the number of the current bit plane.

```
IppStatus ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K
    (IppiDecodeCBProgrState_JPEG2K* pState, int* pBitPlaneNum);
```

### DecodeCBProgrStep\_JPEG2K

Performs a single step of decoding.

```
IppStatus ippiDecodeCBProgrStep_JPEG2K(const Ipp8u* pSrc, int srcLen,
        IppiDecodeCBProgrState_JPEG2K* pState);
```

## Component Transform Functions

### RCTFwd\_JPEG2K

Performs forward reversible component transformation.

#### Case 1: Operation on pixel-order data

```
IppStatus ippiRCTFwd_JPEG2K_32s_C3P3R(const Ipp32s* pSrc, int srcStep,
        Ipp32s* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
IppStatus ippiRCTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
        IppiSize roiSize);
```

## RCTInv\_JPEG2K

Performs inverse reversible component transformation.

### Case 1: Operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3C3R(const Ipp32s* pSrc[3], int
    srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

## ICTFwd\_JPEG2K

Performs forward irreversible component transformation.

### Case 1: Operation on pixel-order data

```
IppStatus ippiICTFwd_JPEG2K_32f_C3P3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation on planar data

```
IppStatus ippiICTFwd_JPEG2K_16s_P3IR(Ipp16s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

```
IppStatus ippiICTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

```
IppStatus ippiICTFwd_JPEG2K_32f_P3IR(Ipp32f* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

## ICTInv\_JPEG2K

Performs inverse irreversible component transformation.

### Case 1: Operation on planar data

```
IppStatus ippiICTInv_JPEG2K_32f_P3C3R(const Ipp32f* pSrc[3], int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: In-place operation on planar data

```
IppStatus ippiICTInv_JPEG2K_16s_P3IR(Ipp16s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

```
IppStatus ippiICTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

```
IppStatus ippiICTInv_JPEG2K_32f_P3IR(Ipp32f* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

# Video Coding

## General Functions

### HuffmanTableInitAlloc

Allocates memory and initializes structure for table with one-to-one code/value correspondence.

```
IppStatus ippiHuffmanTableInitAlloc_32s(const Ipp32s* pSrcTable,  
    IppVCHuffmanSpec_32s** ppDstSpec);
```

### HuffmanRunLevelTableInitAlloc

Allocates memory and initializes structure for table with one-to-two code/value correspondence.

```
IppStatus ippiHuffmanRunLevelTableInitAlloc_32s(const Ipp32s* pSrcTable,  
    IppVCHuffmanSpec_32s** ppDstSpec);
```

### DecodeHuffmanOne

Decodes one code using specified table and gets one decoded value.

```
IppStatus ippiDecodeHuffmanOne_1u32s(Ipp32u** ppBitStream, Ipp32s*  
    pOffset, Ipp32s* pDst, const IppVCHuffmanSpec_32s *pDecodeTable);
```

### DecodeHuffmanPair

Decodes one code using specified table and gets two decoded values.

```
IppStatus ippiDecodeHuffmanPair_1u16s(Ipp32u **ppBitStream, Ipp32s  
    *pOffset, const IppVCHuffmanSpec_32s *pDecodeTable, Ipp8s *pFirst,  
    Ipp16s *pSecond);
```

### HuffmanTableFree

Frees memory allocated for VLC table.

```
IppStatus ippiHuffmanTableFree_32s(IppVCHuffmanSpec_32s  
    **ppDecodeTable);
```

### MC16x16

Performs motion compensation for predicted 16X16 block.

```
IppStatus ippiMC16x16_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const  
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,  
    Ipp32s mcType, Ipp32s roundControl);
```

## MC16x8

Performs motion compensation for predicted 16X8 block.

```
IppStatus ippMC16x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC8x16

Performs motion compensation for predicted 8X16 block.

```
IppStatus ippMC8x16_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC8x8

Performs motion compensation for predicted 8X8 block.

```
IppStatus ippMC8x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC8x4

Performs motion compensation for predicted 8X4 block.

```
IppStatus ippMC8x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC4x8

Performs motion compensation for predicted 4X8 block.

```
IppStatus ippMC4x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC4x4

Performs motion compensation for predicted 4X4 block.

```
IppStatus ippMC4x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC2x4

Performs motion compensation for predicted 2X4 block.

```
IppStatus ippMC2x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```



## MC4x2

Performs motion compensation for predicted 4X2 block.

```
IppStatus ippiMC4x2_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC2x2

Performs motion compensation for predicted 2X2 block.

```
IppStatus ippiMC2x2_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC16x4

Performs motion compensation for predicted UV 16X4 block.

```
IppStatus ippiMC16x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC16x8UV

Performs motion compensation for predicted UV 16X8 block.

```
IppStatus ippiMC16x8UV_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep,
    Ipp32s mcType, Ipp32s roundControl);
```

## MC16x16B

Performs motion compensation for bi-predicted block.

```
IppStatus ippiMC16x16B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC16x8B

Performs motion compensation for bi-predicted 16X8 block.

```
IppStatus ippiMC16x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC8x16B

Performs motion compensation for bi-predicted 8X16 block.

```
IppStatus ippiMC8x16B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC8x8B

Performs motion compensation for bi-predicted 8X8 block.

```
IppStatus ippiMC8x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC8x4B

Performs motion compensation for bi-predicted 8X4 block.

```
IppStatus ippiMC8x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC4x8B

Performs motion compensation for bi-predicted 4X8 block.

```
IppStatus ippiMC4x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC4x4B

Performs motion compensation for bi-predicted 4X4 block.

```
IppStatus ippiMC4x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC2x4B

Performs motion compensation for bi-predicted 2X4 block.

```
IppStatus ippiMC2x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC4x2B

Performs motion compensation for bi-predicted 4X2 block.

```
IppStatus ippiMC4x2B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC2x2B

Performs motion compensation for bi-predicted 2X2 block.

```
IppStatus ippiMC2x2B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC16x4B

Performs motion compensation for bi-predicted UV 16X4 block.

```
IppStatus ippiMC16x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## MC16x8UVB

Performs motion compensation for bi-predicted 16X8 block.

```
IppStatus ippiMC16x8UVB_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF,  
    Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s  
    mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst,  
    Ipp32s dstStep, Ipp32s roundControl);
```

## Copy8x8, Copy16x16

Copy fixed size block.

```
IppStatus ippiCopy8x8_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep);  
  
IppStatus ippiCopy16x16_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep);
```

**Copy8x4HP,  
Copy8x8HP,  
Copy16x8HP,  
Copy16x16HP**

Copy fixed size blocks with half-pixel accuracy.

```
IppStatus ippiCopy8x4HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
IppStatus ippiCopy8x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
IppStatus ippiCopy16x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
IppStatus ippiCopy16x16HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
```

**InterpolateAverage8x4,  
InterpolateAverage8x8,  
InterpolateAverage16x8,  
InterpolateAverage16x16**

Interpolate source block according to half-pixel offset and average the result with destination block.

```
IppStatus ippiInterpolateAverage8x4_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage8x8_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x8_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x16_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
```

**Add8x8**

Adds two blocks with saturation.

```
IppStatus ippiAdd8x8_16s8u_C1IRS(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep);
```

**Add8x8HP**

Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.

```
IppStatus ippiAdd8x8HP_16s8u_C1RS(const Ipp16s* pSrc1, int src1Step,
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, int acc, int
    rounding);
```

## AddC8x8

Adds a constant to 8x8 block with saturation.

```
IppStatus ippiAddC8x8_16s8u_C1IR(Ipp16s value, Ipp8u* pSrcDst, int
    srcDstStep);
```

## Average8x8, Average16x16

Average two blocks.

```
IppStatus ippiAverage8x8_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep);
IppStatus ippiAverage16x16_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep);
```

## GetDiff16x16

Evaluates difference between current predicted and reference blocks of 16x16 elements.

```
IppStatus ippiGetDiff16x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

## GetDiff16x8

Evaluates difference between current predicted and reference blocks of 16x8 elements.

```
IppStatus ippiGetDiff16x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

## GetDiff8x8

Evaluates difference between current predicted and reference blocks of 8x8 elements.

```
IppStatus ippiGetDiff8x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

## GetDiff8x16

Evaluates difference between current predicted and reference blocks of 8x16 elements.

```
IppStatus ippiGetDiff8x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

### GetDiff8x4

Evaluates difference between current predicted and reference blocks of 8x4 elements.

```
IppStatus ippiGetDiff8x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

### GetDiff4x4

Computes difference between current predicted and reference 4x4 blocks.

```
IppStatus ippiGetDiff4x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s*
    pDstDiff, Ipp32s dstDiffStep, Ipp16s* pDstPredictor, Ipp32s
    dstPredictorStep, Ipp32s mcType, Ipp32s roundControl);
```

### GetDiff16x16B

Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.

```
IppStatus ippiGetDiff16x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB,
    Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### GetDiff16x8B

Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.

```
IppStatus ippiGetDiff16x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB,
    Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### GetDiff8x8B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.

```
IppStatus ippiGetDiff8x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB,
    Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### GetDiff8x16B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.

```
IppStatus ippiGetDiff8x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB,
    Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### GetDiff8x4B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.

```
IppStatus ippiGetDiff8x4B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s
    srcCurStep, const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB,
    Ipp16s* pDstDiff, Ipp32s dstDiffStep, Ipp32s roundControl);
```

### Sub8x8, Sub16x16

Subtract two blocks and store the result in the third block.

```
IppStatus ippiSub8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
IppStatus ippiSub16x16_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

### SubSAD8x8

Subtracts two blocks, stores the result in the third block and computes a sum of absolute differences.

```
IppStatus ippiSubSAD8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step,
    const Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, Ipp32s*
    pSAD);
```

### SqrDiff16x16

Evaluates sum of squares of differences between current and reference 16X16 blocks.

```
IppStatus ippiSqrDiff16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u* pRef, Ipp32s refStep, Ipp32s mcType, Ipp32s* pSqrDiff);
```

### SqrDiff16x16B

Evaluates sum of squares of differences between current bi-predicted 16X16 block and mean of two reference blocks.

```
IppStatus ippiSqrDiff16x16B_8u32s(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u pRefF, Ipp32s refStepF, Ipp32s mcTypeF, const Ipp8u
    pRefB, Ipp32s refStepB, Ipp32s mcTypeB, Ipp32s* pSqrDiff);
```

## SSD8x8

Evaluates sum of squares of differences between current and reference 8X8 blocks.

```
IppStatus ippiSSD8x8_8u32s_C1R(const Ipp8u *srcStep, int srcCurStep,  
    const Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

## SSD4x4

Evaluates sum of squares of differences between current and reference 4X4 blocks.

```
IppStatus ippiSSD4x4_8u32s_C1R(const Ipp8u *srcStep, int srcCurStep,  
    const Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

## VarMean8x8

Evaluates variance and mean of 8X8 block.

```
IppStatus ippiVarMean8x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep,  
    Ipp32s* pVar, Ipp32s* pMean);  
IppStatus ippiVarMean8x8_16s32s_C1R(const Ipp16s* pSrc, Ipp32s srcStep,  
    Ipp32s* pVar, Ipp32s* pMean);
```

## VarMeanDiff16x16

Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.

```
IppStatus ippiVarMeanDiff16x16_8u32s_C1R(const Ipp8u* pSrc, Ipp32s  
    srcStep, const Ipp8u* pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s*  
    pVar, Ipp32s* pMean, Ipp32s mcType);
```

## VarMeanDiff16x8

Evaluates variances and means of two 8x8 blocks of difference between two 16x8 blocks.

```
IppStatus ippiVarMeanDiff16x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s  
    srcStep, const Ipp8u* pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s*  
    pVar, Ipp32s* pMean, Ipp32s mcType);
```

## Variance16x16

Evaluates variance of current block.

```
IppStatus ippiVariance16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep,  
    Ipp32s* pVar);
```

## MeanAbsDev16x16

Evaluates mean absolute deviation for a 16x16 block.

```
IppStatus ippiMeanAbsDev16x16_8u32s_C1R(const Ipp8u *pSrc, int srcStep,  
    Ipp32s *pDst);
```



## EdgesDetect16x16

Detects edges inside 16X16 block.

```
IppStatus ippiEdgesDetect16x16_8u_C1R(const Ipp8u *pSrc, Ipp32u srcStep,
    Ipp8u EdgePelDifference, Ipp8u EdgePelCount, Ipp8u *pRes);
```

## SAD16x16

Evaluates sum of absolute difference between current and reference 16X16 blocks.

```
IppStatus ippiSAD16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u* pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

## SAD16x8

Evaluates sum of absolute difference between current and reference 16X8 blocks.

```
IppStatus ippiSAD16x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep,
    const Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

## SAD8x8

Evaluates sum of absolute difference between current and reference 8X8 blocks.

```
IppStatus ippiSAD8x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep,
    const Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

## SAD4x4

Evaluates sum of absolute difference between current and reference 4X4 blocks.

```
IppStatus ippiSAD4x4_8u32s(const Ipp8u *pSrc, Ipp32s srcStep, const Ipp8u
    *pRef, Ipp32s refStep, Ipp32s *pSAD, Ipp32s mcType);
```

## SAD16x16Blocks8x8

Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.

```
IppStatus ippiSAD16x16Blocks8x8_8u16u(const Ipp8u *pSrc, Ipp32s srcStep,
    const Ipp8u *pRef, Ipp32s refStep, Ipp16u *pDstSAD, Ipp32s mcType);
```

## SAD16x16Blocks4x4

Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.

```
IppStatus ippiSAD16x16Blocks4x4_8u16u(const Ipp8u *pSrc, Ipp32s srcStep,
    Ipp8u *pRef, Ipp32s refStep, Ipp16u *pDstSAD, Ipp32s mcType);
```

## FrameFieldSAD16x16

Calculates SAD between field lines and SAD between frame lines of block 16x16.

```
IppStatus ippiFrameFieldSAD16x16_8u32s_C1R(const Ipp8u *pSrc, int
    srcStep, Ipp32s *pFrameSAD, Ipp32s *pFieldSAD);
IppStatus ippiFrameFieldSAD16x16_16s32s_C1R(const Ipp16s *pSrc, int
    srcStep, Ipp32s *pFrameSAD, Ipp32s *pFieldSAD);
```

## SumsDiff16x16Blocks4x4

Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.

```
IppStatus ippiSumsDiff16x16Blocks4x4_8u16s_C1(Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s*
    pDiff);
```

## SumsDiff8x8Blocks4x4

Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.

```
IppStatus ippiSumsDiff8x8Blocks4x4_8u16s_C1(Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

## ScanInv

Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.

```
ippiScanInv_16s_C1(Ipp16s *pSrc, Ipp16s *pDst, int indxLastNonZero, int
    scan);
```

## ScanFwd

Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.

```
ippiScanFwd_16s_C1(Ipp16s *pSrc, Ipp16s *pDst, int countNonZero, int
    scan);
```

## ZigzagInvClassical\_Compact, ZigzagInvHorizontal\_Compact, ZigzagInvVertical\_Compact

Perform classical, horizontal, or vertical inverse zigzag scan on a block stored in a compact buffer.

```
IppStatus ippiZigzagInvClassical_Compact_16s(const Ipp16s* pSrc, int
    len, Ipp16s* pDst);
IppStatus ippiZigzagInvHorizontal_Compact_16s(const Ipp16s* pSrc, int
    len, Ipp16s* pDst);
```

```
IppStatus ippiZigzagInvVertical_Compact_16s(const Ipp16s* pSrc, int len,
Ipp16s* pDst);
```

### **YCbCr420ToBGR565\_Rotate**

Converts YCbCr image to BGR image and makes 90-degree left rotation of destination image.

```
IppStatus ippiYCbCr420ToBGR565_Rotate_8u16u_P3C3R(const Ipp8u* pSrc[3],
int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### **CbYCr422ToYCbCr420\_Rotate**

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.

```
IppStatus ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(const Ipp8u* pSrc, int
srcStep, IppiSize srcRoi, Ipp8u *pDst[3], int dstStep[3], int
rotation);
IppStatus ippiCbYCr422ToYCbCr420_Rotate_8u_P3R(const Ipp8u* pSrc[3], int
srcStep[3], IppiSize srcRoi, Ipp8u *pDst[3], int dstStep[3], int
rotation);
```

### **ResizeCCRotate**

Creates a low-resolution preview image for high-resolution video or still capture applications.

```
IppStatus ippiResizeCCRotate_8u_C2R(const Ipp8u *pSrc, int srcStep,
IppiSize srcRoi, Ipp16u *pDst, int dstStep, int zoomFactor, int
interpolation, int colorConversion, int rotation);
```

### **DeinterlaceFilterTriangle**

Deinterlaces video plane.

```
IppStatus ippiDeinterlaceFilterTriangle_8u_C1R(const Ipp8u *pSrc, Ipp32s
srcStep, Ipp8u *pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32u
centerWeight, Ipp32u layout);
```

## **MPEG-1 and MPEG-2**

### **ReconstructDCTBlock\_MPEG1**

Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

```
IppStatus ippiReconstructDCTBlock_MPEG1_32s(Ipp32u **ppBitStream, int
*pOffset, const Ipp32s *pDCSizeTable, const Ipp32s *pACTable, Ipp32s*
pScanMatrix, int QP, Ipp16s *pQPMatrix, Ipp16s *pDstBlock, Ipp32s
*pDstSize);
```

### ReconstructDCTBlockIntra\_MPEG1

Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

```
IppStatus ippiReconstructDCTBlockIntra_MPEG1_32s(Ipp32u **ppBitStream,
int *pOffset, const Ipp32s *pDCSizeTable, const Ipp32s *pACTable,
Ipp32s* pScanMatrix, int QP, Ipp16s *pQPMatrix, Ipp16s *pDCPred,
Ipp16s *pDstBlock, Ipp32s *pDstSize);
```

### ReconstructDCTBlock\_MPEG2

Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.

```
IppStatus ippiReconstructDCTBlock_MPEG2_32s(Ipp32u **ppBitStream, int
*pOffset, const IppVCHuffmanSpec_32s *pDCTable, const
IppVCHuffmanSpec_32s *pACTable, Ipp32s *pScanMatrix, int QP, Ipp16s
*pQPMatrix, Ipp16s *pDstBlock, Ipp32s *pDstSize);
```

### ReconstructDCTBlockIntra\_MPEG2

Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.

```
IppStatus ippiReconstructDCTBlockIntra_MPEG2_32s(Ipp32u **ppBitStream,
int *pOffset, const IppVCHuffmanSpec_32s *pDCSizeTable, const
IppVCHuffmanSpec_32s *pACTable, Ipp32s *pScanMatrix, int QP, Ipp16s
*pQPMatrix, Ipp16s *pDCPred, Ipp32s shiftDCVal, Ipp16s *pDstBlock,
Ipp32s *pDstSize);
```

### QuantInvIntra\_MPEG2

Performs inverse quantization for intra frames according to MPEG-2 standard.

```
IppStatus ippiQuantInvIntra_MPEG2_16s_C1I(Ipp16s *pSrcDst, int QP,
Ipp16s *pQPMatrix);
```

### QuantInv\_MPEG2

Performs inverse quantization for non-intra frames according to MPEG-2 standard.

```
IppStatus ippiQuantInv_MPEG2_16s_C1I(Ipp16s *pSrcDst, int QP, Ipp16s
*pQPMatrix);
```

### DCT8x8Inv\_AANTransposed\_16s\_C1R

Performs inverse DCT on pre-transposed block.

```
ippiDCT8x8Inv_AANTransposed_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst,
Ipp32s dstStep, Ipp32s count);
```

### **DCT8x8Inv\_AANTransposed\_16s8u\_C1R**

Performs inverse DCT on pre-transposed block and converts output to unsigned char format.

```
ippiDCT8x8Inv_AANTransposed_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst,  
    Ipp32s dstStep, Ipp32s count);
```

### **DCT8x8Inv\_AANTransposed\_16s\_P2C2R**

Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.

```
ippiDCT8x8Inv_AANTransposed_16s_P2C2R(const Ipp16s* pSrcU, const Ipp16s*  
    pSrcV, Ipp16s* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

### **DCT8x8Inv\_AANTransposed\_16s8u\_P2C2R**

Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.

```
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R(const Ipp16s* pSrcU, const  
    Ipp16s* pSrcV, Ipp8u* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s  
    countV);
```

### **QuantIntra\_MPEG2**

Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.

```
IppStatus ippiQuantIntra_MPEG2_16s_C1I(Ipp16s *pSrcDst, Ipp32s QP, const  
    Ipp32f *pQPMatrix, Ipp32s *pCount);
```

### **Quant\_MPEG2**

Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.

```
IppStatus ippiQuant_MPEG2_16s_C1I(Ipp16s *pSrcDst, Ipp32s QP, const  
    Ipp32f *pQPMatrix, Ipp32s *pCount);
```

### **CreateRLEncodeTable**

Creates Run-Level Encode Table.

```
IppStatus ippiCreateRLEncodeTable (const Ipp32s *pSrcTable,  
    IppVCHuffmanSpec_32s** ppDstSpec);
```

## PutIntraBlock

Encodes, rearranges and puts intra block into bit stream.

```
IppStatus ippiPutIntraBlock(Ipp32u **ppBitStream, int *pOffset, Ipp16s*  
    pSrcBlock, Ipp32s *pDCPred, IppVCHuffmanSpec_32u* pDCTable,  
    IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s EOBLen,  
    Ipp32s EOBCode, Ipp32s count);
```

## PutNonIntraBlock

Encodes, rearranges and puts non-intra block into bit stream.

```
IppStatus ippiPutNonIntraBlock(Ipp32u **ppBitStream, int* pOffset,  
    Ipp16s* pSrcBlock, IppVCHuffmanSpec_32s* pACTable, Ipp32s*  
    pScanMatrix, Ipp32s EOBLen, Ipp32s EOBCode, Ipp32s count);
```

## DV

### InitAllocHuffmanTable\_DV

Allocates memory and initializes table.

```
IppStatus ippiInitAllocHuffmanTable_DV_32u(Ipp32s *pSrcTable1, Ipp32s  
    *pSrcTable2, Ipp32u **ppHuffTable);
```

### HuffmanDecodeSegment\_DV

Decodes and rearranges segment block, multiplies first block element by 128.

```
IppStatus ippiHuffmanDecodeSegment_DV_8u16s(Ipp8u *pStream, Ipp32u  
    *pZigzagTables, Ipp32u *pHuffTable, Ipp16s *pBlock, Ipp32u  
    *pBlockParam);
```

### FreeHuffmanTable\_DV

Frees the memory allocated for VLC table.

```
IppStatus ippiFreeHuffmanTable_DV_32u(Ipp32u *pHuffTable);
```

### QuantInv\_DV

Performs inverse quantization on a block.

```
IppStatus ippiQuantInv_DV_16s_C1I(Ipp16s *pSrcDst, Ipp16s  
    *pDequantTable);
```

### DCT2x4x8Inv

Performs the inverse DCT for block of type 2 (m0=1).

```
IppStatus ippiDCT2x4x8Inv_16s_C1I(Ipp16s *pSrcDst);
```

## DCT2x4x8Frw

Performs DCT for a block of type 2.

```
IppStatus ippiDCT2x4x8Frw_16s_C1I(Ipp16s *pSrcDst);
```

## CountZeros8x8

Evaluates number of zeros in a block.

```
IppStatus ippiCountZeros8x8_16s_C1(Ipp16s *pSrc, Ipp32u* pCount);
```

## YCrCb411ToYCbCr422\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV

Convert five YCrCb411 macroblocks into YCrCb422 macroblocks.

```
IppStatus ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5]);
```

## YCrCb411ToYCbCr422\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV

Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.

```
IppStatus ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc,  
    Ipp8u* pDst, int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R(const  
    Ipp16s* pSrc, Ipp8u* pDst, int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R(const  
    Ipp16s* pSrc, Ipp8u* pDst, int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R(const  
    Ipp16s* pSrc, Ipp8u* pDst, int dstStep);
```

**YCrCb420ToYCbCr422\_5MBDV,  
YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV,  
YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV,  
YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV**

Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.

```
IppStatus ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);
```

**YCrCb422ToYCbCr422\_5MBDV,  
YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV,  
YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV,  
YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV**

Convert five YCrCb422 macroblocks into YCrCb422 macroblocks.

```
IppStatus ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const  
    Ipp16s* pSrc[5], Ipp8u* pDst[5]);
```

## MPEG-4

**Copy8x8QP\_MPEG4,  
Copy16x8QP\_MPEG4,  
Copy16x16QP\_MPEG4**

Copy fixed size blocks with quarter-pixel accuracy.

```
IppStatus ippiCopy8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, int acc, int rounding);  
IppStatus ippiCopy16x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, int acc, int rounding);  
IppStatus ippiCopy16x16QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, int acc, int rounding);
```



### **OBCM8x8HP\_MPEG4, OBCM16x16HP\_MPEG4, OBCM8x8QP\_MPEG4**

Perform overlapped block motion compensation (OBMC) for one luminance block.

```
IppStatus ippiOBMC8x8HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur, const
    IppMotionVector* pMVLeft, const IppMotionVector* pMVRight, const
    IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow, int
    rounding);

IppStatus ippiOBMC16x16HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur, const
    IppMotionVector* pMVLeft, const IppMotionVector* pMVRight, const
    IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow, int
    rounding);

IppStatus ippiOBMC8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppMotionVector* pMVCur, const
    IppMotionVector* pMVLeft, const IppMotionVector* pMVRight, const
    IppMotionVector* pMVAbove, const IppMotionVector* pMVBelow, int
    rounding);
```

### **WarpInit\_MPEG4**

Initializes IppiWarpSpec\_MPEG4 structure for further usage in GMC or Sprite reconstruction.

```
IppStatus ippiWarpInit_MPEG4(IppiWarpSpec_MPEG4 *pSpec, const int *pDU,
    const int *pDV, int numWarpingPoints, int spriteType, int
    warpingAccuracy, int roundingType, int quarterSample, int fcode,
    const IppiRect* pSpriteRect, const IppiRect* pVopRect);
```

### **WarpGetSize\_MPEG4**

Returns size of IppiWarpSpec\_MPEG4 structure.

```
IppStatus ippiWarpGetSize_MPEG4(int* pSpecSize);
```

### **WarpLuma\_MPEG4**

Warps arbitrary rectangular luminance region.

```
IppStatus ippiWarpLuma_MPEG4_8u_C1R(const Ipp8u* pSrcY, int srcStepY,
    Ipp8u* pDstY, int dstStepY, const IppiRect* pDstRect, const
    IppiWarpSpec_MPEG4* pSpec);
```

### **WarpChroma\_MPEG4**

Warps arbitrary rectangular chrominance region.

```
IppStatus ippiWarpChroma_MPEG4_8u_P2R(const Ipp8u *pSrcCb, int
    srcStepCb, const Ipp8u *pSrcCr, int srcStepCr, Ipp8u* pDstCb, int
    dstStepCb, Ipp8u* pDstCr, int dstStepCr, const IppiRect* pDstRect,
    const IppiWarpSpec_MPEG4 *pSpec);
```

### CalcGlobalMV\_MPEG4

Calculates Global Motion Vector for one macroblock.

```
IppStatus ippiCalcGlobalMV_MPEG4(int xOffset, int yOffset,  
    IppMotionVector* pGMV, const IppiWarpSpec_MPEG4* pSpec);
```

### ChangeSpriteBrightness\_MPEG4

Change brightness after sprite warping.

```
IppStatus ippiChangeSpriteBrightness_MPEG4_8u_C1IR(const Ipp8u *pSrcDst,  
    int srcDstStep, int width, int height, int brightnessChangeFactor);
```

### DecodePadMV\_PVOP\_MPEG4

Decodes and pads four motion vectors of the non-intra macroblock in P-VOP.

```
IppStatus ippiDecodePadMV_PVOP_MPEG4(Ipp8u** ppBitStream, int*  
    pBitOffset, IppMotionVector* pSrcMVLeftMB, IppMotionVector*  
    pSrcMVUpperMB, IppMotionVector* pSrcMVUpperRightMB, IppMotionVector*  
    pDstMVCurMB, Ipp8u* pTranspLeftMB, Ipp8u* pTranspUpperMB, Ipp8u*  
    pTranspUpperRightMB, Ipp8u* pTranspCurMB, int fCodeForward,  
    IppMacroblockType MBType);
```

### DecodeMV\_BVOP\_Forward\_MPEG4

Decodes motion vector of the macroblock in B-VOP forward mode.

```
IppStatus ippiDecodeMV_BVOP_Forward_MPEG4(Ipp8u** ppBitStream, int*  
    pBitOffset, IppMotionVector* pSrcDstMVF, int fCodeForward);
```

### DecodeMV\_BVOP\_Backward\_MPEG4

Decodes motion vector of the macroblock in B-VOP backward mode.

```
IppStatus ippiDecodeMV_BVOP_Backward_MPEG4(Ipp8u** ppBitStream, int*  
    pBitOffset, IppMotionVector* pSrcDstMVB, int fCodeBackward);
```

### DecodeMV\_BVOP\_Interpolate\_MPEG4

Decodes motion vector of the macroblock in B-VOP interpolate mode.

```
IppStatus ippiDecodeMV_BVOP_Interpolate_MPEG4(Ipp8u** ppBitStream, int*  
    pBitOffset, IppMotionVector* pSrcDstMVF, IppMotionVector* pSrcDstMVB,  
    int fCodeForward, int fCodeBackward);
```

### DecodeMV\_BVOP\_Direct\_MPEG4

Decodes motion vectors of the macroblock in B-VOP using direct mode.

```
IppStatus ippiDecodeMV_BVOP_Direct_MPEG4(Ipp8u** ppBitStream, int*  
    pBitOffset, const IppMotionVector* pSrcMV, IppMotionVector* pDstMVF,  
    IppMotionVector* pDstMVB, Ipp8u* pTranspSrcMB, int TRB, int TRD);
```

## DecodeMV\_BVOP\_DirectSkip\_MPEG4

Decodes motion vectors of the macroblock in B-VOP using direct mode when the current macroblock is skipped.

```
IppStatus ippiDecodeMV_BVOP_DirectSkip_MPEG4 (const IppMotionVector*
    pSrcMV, IppMotionVector* pDstMVF, IppMotionVector* pDstMVB, Ipp8u*
    pTranspSrcMB, int TRB, int TRD);
```

## LimitMVToRect\_MPEG4

Limits the motion vector of current block/macroblock into the extended bounding rectangle.

```
IppStatus ippiLimitMVToRect_MPEG4 (const IppMotionVector* pSrcMV,
    IppMotionVector* pDstMV, IppiRect* pRectVOPRef, int xcoord, int
    ycoord, int size);
```

## PredictReconCoefIntra\_MPEG4

Performs adaptive DC/AC coefficient prediction for an intra block.

```
IppStatus ippiPredictReconCoefIntra_MPEG4_16s (Ipp16s* pSrcDst, Ipp16s*
    pPredBufRow, Ipp16s* pPredBufCol, int curQP, int predQP, int predDir,
    int ACPredFlag, IppVideoComponent videoComp);
```

## PadMBHorizontal\_MPEG4

Performs horizontal extended padding process on exterior macroblock immediately next to boundary macroblocks.

```
IppStatus ippiPadMBHorizontal_MPEG4_8u (const Ipp8u* pSrcY, const Ipp8u*
    pSrcCb, const Ipp8u* pSrcCr, const Ipp8u* pSrcA, Ipp8u* pDstY, Ipp8u*
    pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

## PadMBVertical\_MPEG4

Performs vertical extended padding process on exterior macroblock immediately next to boundary macroblocks.

```
IppStatus ippiPadMBVertical_MPEG4_8u (const Ipp8u* pSrcY, const Ipp8u*
    pSrcCb, const Ipp8u* pSrcCr, const Ipp8u* pSrcA, Ipp8u* pDstY, Ipp8u*
    pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

## PadMBGray\_MPEG4

Fills gray value in exterior macroblock that is not located next to any boundary macroblocks.

```
IppStatus ippiPadMBGray_MPEG4_8u (Ipp8u grayVal, Ipp8u* pDstY, Ipp8u*
    pDstCb, Ipp8u* pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

**PadCurrent\_16x16\_MPEG4,  
PadCurrent\_8x8\_MPEG4**

Perform horizontal and vertical repetitive padding process on luminance/alpha macroblock or chrominance block.

```
IppStatus ippiPadCurrent_16x16_MPEG4_8u_I(Ipp8u* pSrcDst, int step,  
    const Ipp8u* pBAB);  
IppStatus ippiPadCurrent_8x8_MPEG4_8u_I(Ipp8u* pSrcDst, int step, const  
    Ipp8u* pBAB);
```

**PadMV\_MPEG4**

Performs vector padding for a non-transparent macroblock.

```
IppStatus ippiPadMV_MPEG4 (IppMotionVector* pSrcDstMV, Ipp8u* pTransp);
```

**QuantInvIntraInit\_MPEG4,  
QuantInvInterInit\_MPEG4**

Initialize specification structures.

```
IppStatus ippiQuantInvIntraInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantInvIntraSpec_MPEG4* pSpec, int bitsPerPixel);  
IppStatus ippiQuantInvInterInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantInvInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

**QuantInvIntraGetSize\_MPEG4,  
QuantInvInterGetSize\_MPEG4**

Return size of specification structures.

```
IppStatus ippiQuantInvIntraGetSize_MPEG4(int* pSpecSize);  
IppStatus ippiQuantInvInterGetSize_MPEG4(int* pSpecSize);
```

**QuantInvIntra\_MPEG4,  
QuantInvInter\_MPEG4**

Perform inverse quantization on intra/inter coded block.

```
IppStatus ippiQuantInvIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, int  
    indxLastNonZero, const IppiQuantIntraSpec_MPEG4* pSpec, int QP, int  
    blockType);  
IppStatus ippiQuantInvInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, int  
    indxLastNonZero, const IppiQuantInterSpec_MPEG4* pSpec, int QP);  
IppStatus ippiQuantInvIntra_MPEG4_16s_I (Ipp16s* pSrcDst, int QP, const  
    Ipp8u* pQPMatrix, IppVideoComponent videoComp);  
IppStatus ippiQuantInvInter_MPEG4_16s_I (Ipp16s* pSrcDst, int QP, const  
    Ipp8u* pQPMatrix);
```

### DecodeDCIntra\_MPEG4

Decodes one DC coefficient for intra coded block.

```
IppStatus ippiDecodeDCIntra_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pDC, int blockType);
```

### DecodeCoeffsIntra\_MPEG4

Decodes DCT coefficients for intra coded block.

```
IppStatus ippiDecodeCoeffsIntra_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int rvlcFlag,
    int noDCFlag, int scan);
```

### DecodeCoeffsIntraRVLCBack\_MPEG4

Decodes DCT coefficients in backward direction for intra coded block using RVLC.

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u **ppBitStream,
    int *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int
    noDCFlag);
```

### DecodeCoeffsInter\_MPEG4

Decodes DCT coefficients for inter coded block.

```
IppStatus ippiDecodeCoeffsInter_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int rvlcFlagint,
    int scan);
```

### DecodeCoeffsInterRVLCBack\_MPEG4

Decodes DCT coefficients in backward direction for inter coded block using RVLC.

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u **ppBitStream,
    int *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero);
```

### ReconstructCoeffsInter\_MPEG4

Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.

```
IppStatus ippiReconstructCoeffsInter_MPEG4_1u16s(Ipp8u** ppBitStream,
    int* pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int
    rvlcFlag, int scan, const IppiQuantInvInterSpec_MPEG4*
    pQuantInvInterSpec, int QP);
```

### **DecodeVLCZigzag\_IntraDCVLC\_MPEG4, DecodeVLCZigzag\_IntraACVLC\_MPEG4**

Perform VLC decoding and inverse zigzag scan for one intra coded block.

```
IppStatus ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s( Ipp8u**  
    ppBitStream, int* pBitOffset, Ipp16s* pDst, int predDir,  
    IppVideoComponent videoComp);  
  
IppStatus ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s(Ipp8u**  
    ppBitStream, int* pBitOffset, Ipp16s* pDst, int predDir);
```

### **DecodeVLCZigzag\_Inter\_MPEG4**

Performs VLC decoding and inverse zigzag scan for one inter coded block.

```
IppStatus ippiDecodeVLCZigzag_Inter_MPEG4_1u16s(Ipp8u** ppBitStream,  
    int* pBitOffset, Ipp16s* pDst);
```

### **DecodeBlockCoef\_Intra\_MPEG4**

Decodes the INTRA block coefficients.

```
IppStatus ippiDecodeBlockCoef_Intra_MPEG4_1u8u(Ipp8u** ppBitStream, int*  
    pBitOffset, Ipp8u* pDst, int step, Ipp16s* pCoefBufRow, Ipp16s*  
    pCoefBufCol, Ipp8u curQP, Ipp8u* pQPBuf, const Ipp8u* pQPMatrix, int  
    blockIndex, int intraDCVLC, int ACPredFlag);
```

### **DecodeBlockCoef\_Inter\_MPEG4**

Decodes the INTER block coefficients.

```
IppStatus ippiDecodeBlockCoef_Inter_MPEG4_1u16s(Ipp8u** ppBitStream,  
    int* pBitOffset, Ipp16s* pDst, int QP, const Ipp8u* pQPMatrix);
```

### **DecodeBlockCoef\_IntraDCOnly\_MPEG4**

Decodes the INTRA block with only DC coefficient.

```
IppStatus ippiDecodeBlockCoef_IntraDCOnly_MPEG4_1u8u(Ipp8u**  
    ppBitStream, int* pBitOffset, Ipp8u* pDst, int step, Ipp16s*  
    pCoefBufRow, Ipp16s* pCoefBufCol, Ipp8u curQp, Ipp8u* pQpBuf, const  
    Ipp8u* pQPMatrix, int blockIndex, int IntraDCVLC, int ACPredFlag);
```

### **FilterDeblocking8x8HorEdge\_MPEG4, FilterDeblocking8x8VerEdge\_MPEG4**

Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.

```
IppStatus ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst,  
    int step, int QP, int THR1, int THR2);  
  
IppStatus ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst,  
    int step, int QP, int THR1, int THR2);
```

### FilterDeringingThreshold\_MPEG4

Computes threshold values for the deringing filtering through a macroblock.

```
IppStatus ippiFilterDeringingThreshold_MPEG4_8u_P3R(Ipp8u* pSrcY, int
    stepY, const Ipp8u* pSrcCb, int stepCb, const Ipp8u* pSrcCr, int
    stepCr, int threshold[6]);
```

### FilterDeringingSmooth8x8\_MPEG4

Performs deringing filtering of a block.

```
IppStatus ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R(Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, int QP, int threshold);
```

### DecodeCAEIntraH\_MPEG4, DecodeCAEIntraV\_MPEG4

Perform Context Arithmetic Code decoding in the intra macroblock.

```
IppStatus ippiDecodeCAEIntraH_MPEG4_1u8u(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp8u* pBinarySrcDst, int step, int blockSize);
IppStatus ippiDecodeCAEIntraV_MPEG4_1u8u(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp8u* pBinarySrcDst, int step, int blockSize);
```

### DecodeCAEInterH\_MPEG4, DecodeCAEInterV\_MPEG4

Perform Context Arithmetic Code decoding in the inter macroblock.

```
IppStatus ippiDecodeCAEInterH_MPEG4_1u8u(Ipp8u** ppBitStream, int*
    pBitOffset, const Ipp8u* pBinarySrcPred, int offsetPred, Ipp8u*
    pBinarySrcDst, int step, int blockSize);
IppStatus ippiDecodeCAEInterV_MPEG4_1u8u(Ipp8u** ppBitStream, int*
    pBitOffset, const Ipp8u* pBinarySrcPred, int offsetPred, Ipp8u*
    pBinarySrcDst, int step, int blockSize);
```

### DecodeMVS\_MPEG4

Decodes motion vectors of shape according to specification.

```
IppStatus ippiDecodeMVS_MPEG4 (Ipp8u** ppBitStream, int* pBitOffset,
    IppMotionVector* pSrcDstMVS, const Ipp8u* pSrcBABMode, int
    stepBABMode, const IppMotionVector* pSrcMVLeftMB, const
    IppMotionVector* pSrcMVUpperMB, const IppMotionVector*
    pSrcMVUpperRightMB, const Ipp8u* pTranspLeftMB, const Ipp8u*
    pTranspUpperMB, const Ipp8u* pTranspUpperRightMB, int stepBM, int
    predFlag);
```

### **PadMBPartial\_MPEG4**

Performs general padding if the current macroblock is partial.

```
IppStatus ippiPadMBPartial_MPEG4_8u_P4R(const Ipp8u* pSrcBAB, const
    Ipp32u* pSrcTrasptMBLeft, Ipp8u* pSrcDstCurrY, Ipp8u* pSrcDstCurrCb,
    Ipp8u* pSrcDstCurrCr, Ipp8u* pSrcDstCurrA, Ipp8u* pSrcDstPadded, int
    iMBX, int iMBY, int stepYA, int stepCbCr, int stepBinary);
```

### **PadMBTransparent\_MPEG4**

Performs general padding if the current macroblock is transparent.

```
IppStatus ippiPadMBTransparent_MPEG4_8u_P4R(const Ipp32s*
    pSrcTrasptMBLeft, Ipp8u* pSrcDstCurrY, Ipp8u* pSrcDstCurrCb, Ipp8u*
    pSrcDstCurrCr, Ipp8u* pSrcDstCurrA, Ipp8u* pSrcDstPadded, Ipp8u
    grayVal, int iMBX, int iMBY, int iMBXLimit, int iMBYLimit, int
    stepYA, int stepCbCr);
```

### **PadMBOpaque\_MPEG4**

Performs general padding if the current macroblock is opaque.

```
IppStatus ippiPadMBOpaque_MPEG4_8u_P4R(const Ipp32s* pSrcTrasptMBLeft,
    Ipp8u* pSrcDstCurrY, Ipp8u* pSrcDstCurrCb, Ipp8u* pSrcDstCurrCr,
    Ipp8u* pSrcDstCurrA, Ipp8u* pSrcDstPadded, int iMBX, int iMBY, int
    stepYA, int stepCbCr);
```

### **SumNorm\_VOP\_MPEG4**

Performs summation of a block of indicated size.

```
IppStatus ippiSumNorm_VOP_MPEG4_8u16u(Ipp8u* pSrcRef, IppiRect*
    pSrcRefRect, Ipp16u* pDstSumRef, int flag, int step);
```

### **BlockMatch\_Integer\_16x16\_SEA**

Performs 16x16 size block match with sub-region.

```
IppStatus ippiBlockMatch_Integer_16x16_SEA (Ipp8u * pSrcRef, Ipp8u *
    pSrcCurr, Ipp16u * pSrcSumBlk, IppMotionVector * pSrcRefMV,
    IppCoordinate * pSrcPointPos, IppiRect * pSrcRefRect, int *
    pSrcDstminSAD, IppMotionVector * pDstMV, int step, int searchRange,
    int flag);
```

### **MotionEstimation\_16x16\_SEA**

Completes 16X16 size motion estimation using core SEA.

```
IppStatus ippiMotionEstimation_16x16_SEA (Ipp8u * pSrcRef, Ipp8u *
    pSrcReconRef, Ipp16u *pSrcSumBlk, Ipp8u * pSrcCurr, IppiRect *
    pSrcRefRect, IppCoordinate * pSrcPointPos, IppMotionVector
    *pSrcRefMV, IppMotionVector * pDstMV, Ipp8u* pDstPreMbtype, int*
    pDstSAD, int step, int roundControl, int searchRange, int flag);
```



## BlockMatch\_Integer\_16x16\_MVFAST

Performs 16x16 size block match with large and/or small diamond search.

```
IppStatus ippiBlockMatch_Integer_16x16_MVFAST (Ipp8u * pSrcRef, Ipp8u *
    pSrcCurr, IppMotionVector *pSrcCanMV, IppMotionVector *pSrcRefMV,
    IppCoordinate * pSrcPointPos, IppiRect * pSrcRefRect, Ipp8u *
    pSrcSadMap, int * pFlag, int * pSrcDstSAD, IppMotionVector * pDstMV,
    int refStep, int searchRange);
```

## MotionEstimation\_16x16\_MVFAST

Performs fast motion estimation using the MVFAST algorithm.

```
IppStatus ippiMotionEstimation_16x16_MVFAST (Ipp8u * pSrcRef, Ipp8u *
    pSrcReconRef, Ipp8u * pSrcCurr, IppMotionVector *pSrcCanMV,
    IppMotionVector *pSrcRefMV, IppCoordinate * pSrcPointPos, IppiRect *
    pSrcRefRect, Ipp8u * pSrcSADMap, Ipp8u * pSrcBlockSADMap,
    IppMotionVector * pDstMV, Ipp8u *pDstPreMbtype, int* pDstSAD, int
    step, int roundControl, int searchRange);
```

## ComputeTextureErrorBlock\_SAD

Computes texture error of the block with SAD exported.

```
IppStatus ippiComputeTextureErrorBlock_SAD_8u16s(const Ipp8u *pSrc, int
    srcStep, const Ipp8u *pSrcRef, Ipp16s * pDst, int *pDstSAD);
```

## ComputeTextureErrorBlock

Computes texture error of the block.

```
IppStatus ippiComputeTextureErrorBlock_8u16s(const Ipp8u *pSrc, int
    srcStep, const Ipp8u *pSrcRef, Ipp16s *pDst);
```

## QuantIntraInit\_MPEG4, QuantInterInit\_MPEG4

Initialize specification structures.

```
IppStatus ippiQuantIntraInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantIntraSpec_MPEG4* pSpec, int bitsPerPixel);
IppStatus ippiQuantInterInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantIntraSpec_MPEG4* pSpec, int bitsPerPixel);
```

## QuantIntraGetSize\_MPEG4, QuantInterGetSize\_MPEG4

Return size of specification structures.

```
IppStatus ippiQuantIntraGetSize_MPEG4(int* pSpecSize);
IppStatus ippiQuantInterGetSize_MPEG4(int* pSpecSize);
```

## QuantIntra\_MPEG4, QuantInter\_MPEG4

Perform inverse quantization on intra/inter coded block.

```
IppStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, const
    IppiQuantIntraSpec_MPEG4* pSpec, int QP, int* pCountNonZero, int
    blockType);

IppStatus ippiQuantIntra_MPEG4_16s_I(Ipp16s* pSrcDst, Ipp8u QP, int
    blockIndex, const int * pQPMatrix);

IppStatus ippiQuantInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, const
    IppiQuantInterSpec_MPEG4* pSpec, int QP, int* pCountNonZero);

IppStatus ippiQuantInter_MPEG4_16s_I(Ipp16s* pSrcDst, Ipp8u QP, const
    int * pQPMatrix);
```

## EncodeDCIntra\_MPEG4

Encodes one DC coefficient for intra coded block.

```
IppStatus ippiEncodeDCIntra_MPEG4_16slu(Ipp16s DC, Ipp8u **ppBitStream,
    int *pBitOffset, int blockType);
```

## EncodeCoeffsIntra\_MPEG4

Encodes DCT coefficients for intra coded block.

```
IppStatus ippiEncodeCoeffsIntra_MPEG4_16slu(const Ipp16s *pCoeffs, Ipp8u
    **ppBitStream, int *pBitOffset, int countNonZero, int rvlcFlag, int
    noDCFlag, int scan);
```

## EncodeCoeffsInter\_MPEG4

Encodes DCT coefficients for inter coded block.

```
IppStatus ippiEncodeCoeffsInter_MPEG4_16slu(const Ipp16s *pCoeffs, Ipp8u
    **ppBitStream, int *pBitOffset, int *countNonZero, int rvlc, int
    scan);
```

## EncodeVLCZigzag\_IntraDCVLC\_MPEG4, EncodeVLCZigzag\_IntraACVLC\_MPEG4

Perform zigzag scanning and VLC encoding for one intra block.

```
IppStatus ippiEncodeVLCZigzag_IntraDCVLC_MPEG4_16slu(Ipp8u**
    ppBitStream, int* pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u predDir,
    Ipp8u pattern, IppVideoComponent videoComp);

IppStatus ippiEncodeVLCZigzag_IntraACVLC_MPEG4_16slu(Ipp8u**
    ppBitStream, int* pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u predDir,
    Ipp8u pattern);
```

## EncodeVLCZigzag\_Inter\_MPEG4

Performs classical zigzag scanning and VLC encoding for one inter block.

```
IppStatus ippiEncodeVLCZigzag_Inter_MPEG4_16s1u(Ipp8u **ppBitStream,
int* pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u pattern);
```

## TransRecBlockCoef\_inter\_MPEG4

Implements DCT, quantizes DCT coefficients of the inter block, and reconstructs the texture residual in the process.

```
IppStatus ippiTransRecBlockCoef_inter_MPEG4 (Ipp16s *pSrc, Ipp16s *
pDst, Ipp16s * pRec, Ipp8u QP, const int * pQPMatrix);
```

## TransRecBlockCoef\_intra\_MPEG4

Quantizes DCT coefficients, implements AC/DC coefficients prediction of the intra block, and stores them into buffer.

```
IppStatus ippiTransRecBlockCoef_intra_MPEG4 (Ipp8u *pSrc, Ipp16s * pDst,
Ipp8u * pRec, Ipp16s *pPredBufRow, Ipp16s *pPredBufCol, Ipp16s *
pPreACPredict, int *pSumErr, int blockIndex, Ipp8u QP, Ipp8u *pQpBuf,
int srcStep, int dstStep, const int * pQPMatrix);
```

## FindMVPred\_MPEG4

Finds the vector predictor from three candidates.

```
IppStatus ippiFindMVPred_MPEG4 (IppMotionVector* pSrcMVCurMB,
IppMotionVector* pSrcCandMV1, IppMotionVector* pSrcCandMV2,
IppMotionVector* pSrcCandMV3, Ipp8u* pSrcCandTransp1, Ipp8u*
pSrcCandTransp2, Ipp8u* pSrcCandTransp3, Ipp8u* pSrcTranspCurr,
IppMotionVector* pDstMVPred, IppMotionVector* pDstMVPredME, int
blockIndex);
```

## EncodeMV\_MPEG4

Finds the prediction MV and encodes the difference.

```
IppStatus ippiEncodeMV_MPEG4_8u16s(Ipp8u **ppBitStream, int *pBitOffset,
IppMotionVector * pMVCurMB, IppMotionVector * pSrcMVLeftMB,
IppMotionVector * pSrcMVUpperMB, IppMotionVector *
pSrcMVUpperRightMB, Ipp8u * pTranspCurMB, Ipp8u * pTranspLeftMB,
Ipp8u * pTranspUpperMB, Ipp8u* pTranspUpperRightMB, int fCodeForward,
IppMacroblockType MBType);
```

## UpdateRCModel\_MPEG4

Updates the rate control model after encoding one frame.

```
IppStatus ippiUpdateRCModel_MPEG4 (IppMP4RCStatus *pRCMode, Ipp32u
bitsTotalCurr, Ipp32u bitsHeadCurr);
```

## UpdateQP\_MPEG4

Calculates quantization level before encoding the current frame.

```
IppStatus ippiUpdateQP_MPEG4 (IppMP4RCStatus *pRCMode);
```

## H.261

### DecodeCoeffsIntra\_H261

Decodes DCT coefficients for intra coded block.

```
ippiDecodeCoeffsIntra_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset,  
    Ipp16s *pCoef, int *pIndxLastNonZero, int scan);
```

### DecodeCoeffsInter\_H261

Decodes DCT coefficients for inter coded block.

```
ippiDecodeCoeffsInter_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset,  
    Ipp16s *pCoef, int *pIndxLastNonZero, int scan);
```

### ReconstructCoeffsIntra\_H261

Reconstructs DCT coefficients for intra coded block.

```
ippiReconstructCoeffsIntra_H261_1u16s(Ipp8u **ppBitStream, int  
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int QP);
```

### ReconstructCoeffsInter\_H261

Reconstructs DCT coefficients for inter coded block.

```
ippiReconstructCoeffsInter_H261_1u16s(Ipp8u **ppBitStream, int  
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int QP);
```

### EncodeCoeffsIntra\_H261

Encodes and puts quantized DCT coefficients for intra coded block into bitstream.

```
IppStatus ippiEncodeCoeffsIntra_H261_16s1u(Ipp16s *pQCoef, Ipp8u  
    **ppBitStream, int *pBitOffset, int countNonZero, int scan);
```

### EncodeCoeffsInter\_H261

Encodes and puts quantized DCT coefficients for inter coded block into bitstream.

```
IppStatus ippiEncodeCoeffsInter_H261_16s1u(Ipp16s *pQCoef, Ipp8u  
    **ppBitStream, int *pBitOffset, int countNonZero, int scan);
```

## Filter8x8\_H261

Performs two-dimensional filtering on a block.

```
ippiFilter8x8_H261_8u_C1R(Ipp8u *pSrc, int srcStep, Ipp8u *pDst, int  
    dstStep);
```

## H.263

### DecodeBlockCoef\_Intra\_H263

Decodes the INTRA block coefficients.

```
IppStatus ippiDecodeBlockCoef_Intra_H263_1u8u(Ipp8u** ppBitStream, int  
    pBitOffset, Ipp8u* pDst, int step, int QP);
```

### DecodeBlockCoef\_Inter\_H263

Decodes the INTER block coefficients.

```
IppStatus ippiDecodeBlockCoef_Inter_H263_1u16s(Ipp8u** ppBitStream, int*  
    pBitOffset, Ipp16s* pDst, int QP);
```

### DecodeDCIntra\_H263

Decodes DC coefficient for intra coded block.

```
IppStatus ippiDecodeDCIntra_H263_1u16s(Ipp8u **ppBitStream, int  
    *pBitOffset, Ipp16s *pDC);
```

### DecodeCoeffsIntra\_H263

Decodes AC coefficients for intra coded block.

```
IppStatus ippiDecodeCoeffsIntra_H263_1u16s(Ipp8u **ppBitStream, int  
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int advIntraFlag,  
    int modQuantFlag, int scan);
```

### DecodeCoeffsInter\_H263

Decodes DCT coefficients for inter coded block.

```
IppStatus ippiDecodeCoeffsInter_H263_1u16s(Ipp8u **ppBitStream, int  
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int modQuantFlag,  
    int scan);
```

### QuantInvIntra\_H263

Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.

```
IppStatus ippiQuantInvIntra_H263_16s_C1I(Ipp16s *pSrcDst, int  
    indxLastNonZero, int QP, int advIntraFlag, int modQuantFlag);
```

### QuantInvInter\_H263

Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.

```
IppStatus ippiQuantInvInter_H263_16s_C1I(Ipp16s *pSrcDst, int
    indxLastNonZero, int QP, int modQuantFlag);
```

### AddBackPredPB\_H263

Performs bidirectional prediction for a B-block in a PB-frame.

```
IppStatus ippiAddBackPredPB_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u *pSrcDst, int srcDstStep, int acc);
```

### ExpandFrame\_H263

Expands the frame to the plane.

```
IppStatus ippiExpandFrame_H263_8u(Ipp8u* pSrcDstPlane, int frameWidth,
    int frameHeight, int expandPels, int step);
```

### Resample\_H263

Performs picture resampling defined in terms of picture area corner displacements.

```
IppStatus ippiResample_H263_8u_P3R(const Ipp8u *pSrcY, int srcYStep,
    IppiSize ySrcRoiSize, const Ipp8u *pSrcCb, int srcCbStep, const Ipp8u
    *pSrcCr, int srcCrStep, Ipp8u *pDstY, int dstYStep, IppiSize
    dstYRoiSize, Ipp8u *pDstCb, int dstCbStep, Ipp8u *pDstCr, int
    dstCrStep, IppMotionVector warpParams[4], int wda, int rounding, int
    fillMode, int fillColor[3]);
```

### UpsampleFour\_H263

Performs factor-of-4 picture upsampling.

```
IppStatus ippiUpsampleFour_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u *pDst, int dstStep, int rounding, int
    fillColor);
```

### DownsampleFour\_H263

Performs factor-of-4 picture downsampling.

```
IppStatus ippiDownsampleFour_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u *pDst, int dstStep, int rounding);
```

### UpsampleFour8x8\_H263

Performs factor-of-4 upsampling on an 8x8 block.

```
IppStatus ippiUpsampleFour8x8_H263_16s_C1R(const Ipp16s *pSrc, int
    srcStep, Ipp16s *pDst, int dstStep);
```

### **SpatialInterpolation\_H263**

Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.

```
IppStatus ippiSpatialInterpolation_H263_8u_C1R(const Ipp8u *pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp8u *pDst, int dstStep, int
    interpType);
```

### **FilterBlockBoundaryHorEdge\_H263, FilterBlockBoundaryVerEdge\_H263**

Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.

```
IppStatus ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR(Ipp8u *pSrcDst,
    int srcDstStep);
IppStatus ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR(Ipp8u *pSrcDst,
    int srcDstStep);
```

### **FilterDeblocking8x8HorEdge\_H263, FilterDeblocking8x8VerEdge\_H263**

Perform deblocking filtering of one block edge on the reconstructed frames.

```
IppStatus ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst,
    int srcDstStep, int QP);
IppStatus ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst,
    int srcDstStep, int QP);
```

### **FilterDeblocking16x16HorEdge\_H263, FilterDeblocking16x16VerEdge\_H263**

Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.

```
IppStatus ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR(Ipp8u *pSrcDst,
    int srcDstStep);
IppStatus ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR(Ipp8u *pSrcDst,
    int srcDstStep);
```

### **ReconstructCoeffsIntra\_H263**

Reconstructs DCT coefficients for an intra coded block.

```
IppStatus ippiReconstructCoeffsIntra_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIdxLastNonZero, int cbp, int QP,
    int advIntraFlag, int scan, int modQuantFlag);
```

### ReconstructCoeffsInter\_H263

Reconstructs DCT coefficients for an inter coded block.

```
IppStatus ippiReconstructCoeffsInter_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int QP, int
    modQuantFlag);
```

### EncodeDCIntra\_H263

Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.

```
IppStatus ippiEncodeDCIntra_H263_16slu(Ipp16s qDC, Ipp8u **ppBitStream,
    int *pBitOffset);
```

### EncodeCoeffsIntra\_H263

Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.

```
IppStatus ippiEncodeCoeffsIntra_H263_16slu(Ipp16s *pQCoef, Ipp8u
    **ppBitStream, int *pBitOffset, int countNonZero, int advIntraFlag,
    int modQuantFlag, int scan);
```

### EncodeCoeffsInter\_H263

Encodes and puts quantized DCT coefficients for inter coded block into bitstream.

```
IppStatus ippiEncodeCoeffsInter_H263_16slu(Ipp16s *pQCoef, Ipp8u
    **ppBitStream, int *pBitOffset, int countNonZero, int modQuantFlag,
    int scan);
```

### QuantIntra\_H263

Performs quantization on an intra coded block.

```
IppStatus ippiQuantIntra_H263_16s_C1I(Ipp16s *pSrcDst, int QP, int
    *pCountNonZero, int advIntraFlag, int modQuantFlag);
```

### QuantInter\_H263

Performs quantization on an inter coded block.

```
IppStatus ippiQuantInter_H263_16s_C1I(Ipp16s *pSrcDst, int QP, int
    *pCountNonZero, int modQuantFlag);
```



## H.264

### DecodeCAVLCCoeffs\_H264

Decodes any non-Chroma DC coefficients CAVLC coded.

```
IppStatus ippiDecodeCAVLCCoeffs_H264_1u16s(Ipp32u **ppBitStream, Ipp32s
    *pBitOffset, Ipp16s *pNumCoeff, Ipp16s **ppDstCoeffs, const Ipp32u
    uVLCSelect, const Ipp16s uMaxNumCoeff, const Ipp32s
    **ppTblCoeffToken, Ipp32s **ppTblTotalZeros, Ipp32s **ppTblRunBefore,
    Ipp32s *pScanMatrix);
```

### DecodeCAVLCChromaDcCoeffs\_H264

Decodes Chroma DC coefficients CAVLC coded.

```
IppStatus ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s(Ipp32u **ppBitStream,
    Ipp32s *pBitOffset, Ipp16s *pNumCoeff, Ipp16s **ppDstCoeffs, const
    Ipp32 *pTblCoeffToken, const Ipp32s **ppTblTotalZerosCR, const Ipp32s
    **ppTblRunBefore);
```

### DecodeExpGolombOne\_H264

Decodes one Exp-Golomb code according to 9.1 H.264 standard.

```
IppStatus ippiDecodeExpGolombOne_H264_1u16s(Ipp32u **ppBitStream, Ipp32s
    *pBitOffset, Ipp16s *pDst, Ipp8u isSigned);
```

### TransformDequantLumaDC\_H264

Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.

```
IppStatus ippiTransformDequantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst,
    Ipp32s QP);
```

### TransformDequantChromaDC\_H264

Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.

```
IppStatus ippiTransformDequantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst,
    Ipp32s QP);
```

### DequantTransformResidual\_H264

Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.

```
IppStatus ippiDequantTransformResidual_H264_16s_C1I(Ipp16s* pSrcDst,
    Ipp32s step, Ipp16s* pDC, Ipp32s AC, Ipp32s QP);
```

### DequantTransformResidualAndAdd\_H264

Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.

```
IppStatus ippiDequantTransformResidualAndAdd_H264_16s_C1I(const Ipp8u*  
    pPred, Ipp16s* pSrcDst, Ipp16s* pDC, Ipp8u* pDst, Ipp32s PredStep,  
    Ipp32s DstStep, Ipp32s QP, Ipp32s AC);
```

### TransformPrediction\_H264

Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.

```
IppStatus ippiTransformPrediction_H264_8u16s_C1(Ipp8u *pSrc, Ipp32s  
    step, Ipp16s *pDst);
```

### DequantTransformResidual\_SISP\_H264

Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.

```
IppStatus ippiDequantTransformResidual_SISP_H264_16s_C1I(Ipp16s*  
    pSrcDst, Ipp16s* pPredictBlock, Ipp16s* pDC, Ipp32s AC, Ipp32s qp,  
    Ipp32s qs, Ipp32s Switch);
```

### TransformDequantChromaDC\_SISP\_H264

Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.

```
IppStatus ippiTransformDequantChromaDC_SISP_H264_16s_C1I(Ipp16s*  
    pSrcDst, Ipp16s* pDCPredict, Ipp32s qp, Ipp32s qs, Ipp32s Switch);
```

### PredictIntra\_4x4\_H264

Performs intra prediction for a 4x4 luma component.

```
IppStatus ippiPredictIntra_4x4_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s  
    srcdstStep, IppIntra4x4PredMode predMode, Ipp32s availability);
```

### PredictIntra\_16x16\_H264

Performs intra prediction for a 16x16 luma component.

```
IppStatus ippiPredictIntra_16x16_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s  
    srcdstStep, IppIntra16x16PredMode predMode, Ipp32s availability);
```

### PredictIntraChroma8x8\_H264

Performs intra prediction for a 8x8 chroma component.

```
IppStatus ippiPredictIntraChroma8x8_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s  
    srcdstStep, IppIntraChroma8x8PredMode predMode, Ipp32s availability);
```

## ExpandPlane\_H264

Expands plane.

```
IppStatus ippiExpandPlane_H264_8u_C1R(Ipp8u *StartPtr, Ipp32u
    uFrameWidth, Ipp32u uFrameHeight, Ipp32u uPitch, Ipp32u uPels,
    IppvcFrameFieldFlag uFrameFieldFlag);
```

## InterpolateLuma\_H264

Performs interpolation for motion estimation of the luma component.

```
IppStatus ippiInterpolateLuma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize
    roiSize);
```

## InterpolateLumaTop\_H264

Performs interpolation for motion estimation of the luma component at the frame top boundary.

```
IppStatus ippiInterpolateLumaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
```

## InterpolateLumaBottom\_H264

Performs interpolation for motion estimation of the luma component at the frame bottom boundary.

```
IppStatus ippiInterpolateLumaBottom_H264_8u_C1R(const Ipp8u* pSrc,
    Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy,
    Ipp32s outPixels, IppiSize roiSize);
```

## InterpolateChroma\_H264

Performs interpolation for motion estimation of the chroma component.

```
IppStatus ippiInterpolateChroma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize
    roiSize);
```

## InterpolateChromaTop\_H264

Performs interpolation for motion estimation of the chroma component at the frame top boundary.

```
IppStatus ippiInterpolateChromaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
```

## InterpolateChromaBottom\_H264

Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.

```
IppStatus ippiInterpolateChromaBottom_H264_8u_C1R(const Ipp8u* pSrc,
    Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy,
    Ipp32s outPixels, IppiSize roiSize);
```

## InterpolateBlock\_H264

Calculates average value for each source pair values in block.

```
IppStatus ippiInterpolateBlock_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u *pSrc2,
    Ipp8u *pDst, Ipp32u width, Ipp32u height, Ipp32u pitch);
```

## WeightedAverage\_H264

Averages two blocks with weights.

```
IppStatus ippiWeightedAverage_H264_8u_C1IR(const Ipp8u* pSrc1, Ipp8u*
    pSrc2Dst, Ipp32s srcDstStep, Ipp32s weight1, Ipp32s weight2, Ipp32s
    shift, Ipp32s offset, IppiSize roiSize);
```

## UniDirWeightBlock\_H264

Weights source block.

```
IppStatus ippiUniDirWeightBlock_H264_8u_C1IR(Ipp8u *pSrcDst, Ipp32u
    srcDstStep, Ipp32u ulog2wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize
    roi);
```

## BiDirWeightBlock\_H264

Averages two blocks with two weights and two offsets.

```
IppStatus ippiBiDirWeightBlock_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u *pSrc2,
    Ipp8u *pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32u ulog2wd, Ipp32s
    iWeight1, Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s iOffset2, IppiSize
    roi);
```

## BiDirWeightBlockImplicit\_H264

Averages two blocks with weights if ulog2wd is equal to 5.

```
IppStatus ippiBiDirWeightBlockImplicit_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u
    *pSrc2, Ipp8u *pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32s iWeight1,
    Ipp32s iWeight2, IppiSize roi);
```

## ReconstructChromaInterMB\_H264

Reconstructs Inter Chroma macroblock.

```
IppStatus ippiReconstructChromaInterMB_H264_16s8u_P2R(Ipp16s
    **ppSrcCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, const
    Ipp32u srcDstStep, const Ipp32u cbp4x4, const Ipp32s ChromaQP);
```

## ReconstructChromaIntraHalvesMB\_H264

Reconstructs two halves of Intra Chroma macroblock.

```
IppStatus ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R(Ipp16s
    **ppSrcCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u
    cbp4x4, Ipp32u ChromaQP, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom);
```

## ReconstructChromaIntraMB\_H264

Reconstructs Intra Chroma macroblock.

```
IppStatus ippiReconstructChromaIntraMB_H264_16s8u_P2R(Ipp16s
    **ppSrcCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, const
    Ipp32u srcDstUVStep, const IppIntraChromaPredMode_H264
    intra_chroma_mode, const Ipp32u cbp4x4, const Ipp32u ChromaQP, const
    Ipp8u edge_type);
```

## ReconstructChromaIntraHalves4x4MB\_H264

Reconstructs two independent halves of 4X4 Intra Chroma macroblock for high profile.

```
IppStatus ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u
    cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeTypeTop, Ipp8u
    edgeTypeBottom, Ipp16s *pQuantTableU, Ipp16s *pQuantTableV, Ipp8u
    bypassFlag);
```

## ReconstructChromaIntra4x4MB\_H264

Reconstructs 4x4 Inter Chroma macroblock for high profile.

```
IppStatus ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u
    srcDstUVStep, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV,
    Ipp16s *pQuantTableU, Ipp16s *pQuantTableV, Ipp8u bypassFlag);
```

## ReconstructLumaInterMB\_H264

Reconstructs Inter Luma macroblock.

```
IppStatus ippiReconstructLumaInterMB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff,
    Ipp8u *pSrcDstYPlane, const Ipp32u srcDstYStep, const Ipp32u cbp4x4,
    const Ipp32s QP);
```

### ReconstructLumaIntraHalfMB\_H264

Reconstructs half of Intra Luma macroblock.

```
IppStatus ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R(Ipp16s
    **ppSrcCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,
    IppIntra4x4PredMode_H264 *pMBIntraTypes, Ipp32u cbp4x2, Ipp32u QP,
    Ipp8u edgeType);
```

### ReconstructLumaIntraMB\_H264

Reconstructs Intra Luma macroblock.

```
IppStatus ippiReconstructLumaIntraMB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff,
    Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep, const
    IppIntra4x4PredMode_H264 *pMBIntraTypes, const Ipp32u cbp4x4, const
    Ipp32u QP, const Ipp8u edgeType);
```

### ReconstructLumaInter4x4MB\_H264

Reconstructs 4x4 Inter Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u
    cbp4x4, Ipp32s QP, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaIntraHalf4x4MB\_H264

Reconstructs half of 4x4 Intra Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,,
    IppIntra4x4PredMode_H264 *pMBIntraTypes, Ipp32u cbp4x2, Ipp32s QP,
    Ipp8u edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaIntra4x4MB\_H264

Reconstructs 4x4 Intra Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,,
    IppIntra4x4PredMode_H264 *pMBIntraTypes, Ipp32u cbp4x4, Ipp32s QP,
    Ipp8u edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaInter8x8MB\_H264

Reconstructs 8x8 Inter Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaInter8x8MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u
    cbp8x8, Ipp32s QP, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaIntraHalf8x8MB\_H264

Reconstructs half of 8x8 Intra Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,
    IppIntra8x8PredMode_H264 *pMBOIntraTypes, Ipp32u cbp8x2, Ipp32s QP,
    Ipp8u edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaIntra8x8MB\_H264

Reconstructs 8x8 Intra Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,
    IppIntra8x8PredMode_H264 *pMBOIntraTypes, Ipp32u cbp8x2, Ipp32s QP,
    Ipp8u edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### ReconstructLumaIntra16x16MB\_H264

Reconstructs Intra 16x16 Luma macroblock.

```
IppStatus ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R(Ipp16s
    **ppSrcCoeff, Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, const
    IppIntra16x16PredMode_H264 intraLumaMode, const Ipp32u cbp4x4,
    const Ipp32u QP, const Ipp8u edge_type);
```

### ReconstructLumaIntra\_16x16MB\_H264

Reconstructs 16x16 Intra Luma macroblock for high profile.

```
IppStatus ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep,
    IppIntra16x16PredMode_H264 intraLumaMode, Ipp32u cbp4x4, Ipp32u QP,
    Ipp8u edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### FilterDeblockingLuma\_VerEdge\_H264

Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.

```
IppStatus ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR( Ipp8u*
    pSrcDst, Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u*
    pThresholds, Ipp8u* pBS);
```

### FilterDeblockingLuma\_VerEdge\_MBAFF\_H264

Performs deblocking filtering on the external vertical edges of half of 16x16 luma macroblock.

```
IppStatus ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR( Ipp8u*
    pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, Ipp8u*
    pThresholds, Ipp8u* pBS);
```

### FilterDeblockingLuma\_HorEdge\_H264

Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.

```
IppStatus ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR( Ipp8u*  
    pSrcDst, Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u*  
    pThresholds, Ipp8u* pBS);
```

### FilterDeblockingChroma\_VerEdge\_H264

Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.

```
IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR( Ipp8u*  
    pSrcDst, Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u*  
    pThresholds, Ipp8u* pBS);
```

### FilterDeblockingChroma\_VerEdge\_MBAFF\_H264

Performs deblocking filtering on the vertical edges of half of 8x8 chroma macroblock.

```
IppStatus ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR( Ipp8u*  
    pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, Ipp8u*  
    pThresholds, Ipp8u* pBS);
```

### FilterDeblockingChroma\_HorEdge\_H264

Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.

```
IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR( Ipp8u*  
    pSrcDst, Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u*  
    pThresholds, Ipp8u* pBS);
```

### TransformQuantChromaDC\_H264

Performs forward transform and quantization for 2x2 DC Chroma blocks.

```
IppStatus ippiTransformQuantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst,  
    Ipp16s* pTBlock, Ipp32s QPChroma, Ipp8s* NumLevels, Ipp8u Intra,  
    Ipp8u NeedTransform);
```

### TransformQuantLumaDC\_H264

Performs forward transform and quantization for 4x4 DC Luma blocks.

```
IppStatus ippiTransformQuantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s*  
    pTBlock, Ipp32s QP, Ipp8s* NumLevels, Ipp8u NeedTransform, Ipp16s*  
    pScanMatrix, Ipp8u* LastCoeff);
```

### TransformQuantResidual\_H264

Performs forward transform and quantization for 4x4 residual blocks.

```
IppStatus ippiTransformQuantResidual_H264_16s_C1I(Ipp16s* pSrcDst,  
    Ipp32s QP, Ipp8s* NumLevels, Ipp8u Intra, Ipp16s* pScanMatrix, Ipp8u*  
    LastCoeff);
```



### TransformLuma8x8Fwd\_H264

Performs forward 8X8 transform for 8X8 Luma blocks without normalisation.

```
IppStatus ippiTransformLuma8x8Fwd_H264_16s_C1I(Ipp16s* pSrcDst);
```

### QuantLuma8x8\_H264

Performs quantization for 8X8 Luma block coefficients including 8X8 transform normalization.

```
IppStatus ippiQuantLuma8x8_H264_16s_C1(const Ipp16s *pSrc, Ipp16s *pDst,  
    int Qp6, int Intra, const Ipp16s *pScanMatrix, const Ipp16s  
    *pScaleLevels, int *pNumLevels, int *pLastCoeff);
```

### GenScaleLevel8x8\_H264

Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8X8 forward and inverse transform.

```
IppStatus ippiGenScaleLevel8x8_H264_8u16s_D2(const Ipp8u  
    *pSrcInvScaleMatrix, int SrcStep, Ipp16s *pDstInvScaleMatrix, Ipp16s  
    *pDstScaleMatrix, int QpRem);
```

### EncodeCoeffsCAVLC\_H264

Calculates characteristics of 4x4 block for CAVLC encoding.

```
IppStatus ippiEncodeCoeffsCAVLC_H264_16s(Ipp16s* pSrc, Ipp8u AC, Ipp32u  
    *pScanMatrix, Ipp8u Count, Ipp8u *Traling_One, Ipp8u  
    *Traling_One_Signs, Ipp8u *NumOutCoeffs, Ipp8u *TotalZeros, Ipp16s  
    *pLevels, Ipp8u *pRuns);
```

### EncodeChromaDcCoeffsCAVLC\_H264

Calculates characteristics of 2x2 Chroma DC block for CAVLC encoding.

```
IppStatus ippiEncodeChromaDcCoeffsCAVLC_H264_16s(Ipp16s* pSrc, Ipp8u  
    *Traling_One, Ipp8u *Traling_One_Signs, Ipp8u *NumOutCoeffs, Ipp8u  
    *TotalZeros, Ipp16s *pLevels, Ipp8u *pRuns);
```

### QuantLuma8x8Inv\_H264

Performs inverse quantization for 8X8 Luma block coefficients including normalization of the subsequent inverse 8X8 transform.

```
IppStatus ippiQuantLuma8x8Inv_H264_16s_C1I(Ipp16s *pSrcDst, int Qp6,  
    const Ipp16s *pInvLevelScale);
```

## **TransformLuma8x8InvAddPred\_H264**

Performs inverse 8X8 transform of 8X8 Luma block coefficients with subsequent intra prediction or motion compensation.

```
IppStatus ippTransformLuma8x8InvAddPred_H264_16s8u_C1R(const Ipp8u  
    *pPred, int PredStep, Ipp16s *pSrc, Ipp8u *pDst, int DstStep);
```